UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA EN INFORMÁTICA

PROYECTO FIN DE CARRERA

Extensión de NCTUns 5.0 para simular el entorno de infraestructura y desarrollo del sistema de representación de indicadores para EVIGEN

Alumno: Sergio García Rueda
Profesor Tutor: José María de Fuentes García-Romero de Tejada

OCTUBRE, 2010

Agradecimientos.

Tengo que dar las gracias a muchas personas que con su apoyo me han dado fuerzas para poder llevar este Proyecto de una manera mucho más amena, sobre todo en los momentos en los que los problemas abordaban y parecía que se andaba por un camino tormentoso que no parecía tener fin.

Mis amigos, que en los momentos más difíciles siempre han estado ahí, y han conseguido hacerme pasar buenos ratos.

Mis hermanos mayores, que siempre me han apoyado y que me han servido de ejemplo para intentar superarme.

Mi tutor Chema, que siempre ha tenido palabras de ánimo hacia mí, y el cual ha puesto mucho empeño en conseguir que este Proyecto llegue a buen fin.

Mi novia, que en ningún momento me ha fallado y que ha estado en cada instante que lo he necesitado a mi lado animándome, y que sin su apoyo, no habría sido capaz de llevar esta carga con la misma buena actitud con la que la he afrontado.

Mis padres, que siempre han confiado en mí, y que se han dedicado durante toda su vida a trabajar para que no me faltase de nada, y que de no ser por ellos, no creo que hubiese llegado al sitio en el que me encuentro actualmente.

Por último, agradecer a todos los futuros lectores del presente trabajo, ya que con el simple hecho de mostrar interés en el mismo, proporcionan un gran apoyo para poder seguir trabajando en un campo de desarrollo que presenta un futuro muy prometedor, y que si no fuese por ellos todo el esfuerzo empleado en la realización de este Proyecto carecería de sentido.

Resumen.

El desarrollo de las tecnologías de la información y las comunicaciones está presentando cada día mayor importancia dentro del ámbito de las redes vehiculares. Debido a esto, aparecen nuevos servicios o aplicaciones destinadas a implantarse en estas redes, con el fin de proporcionar mejoras en seguridad vial, en la actividad de conducción, y soluciones a problemas cuya solución no resulta viable en otros entornos.

No obstante, los costes de implantación de dichas aplicaciones o servicios en estos entornos son muy elevados, debido a la gran cantidad de infraestructura de la que se necesita hacer uso para su desarrollo.

Por este motivo, surge la necesidad de estudiar el funcionamiento y rendimiento de dichos sistemas en estos entornos antes de realizar su implantación, con el objetivo de evitar grandes pérdidas, especialmente económicas, si el mismo no satisface las necesidades para las que ha sido creado.

Debido a esto, surgen los simuladores de redes vehiculares, como NCTUns [Referencias-1], que permiten la simulación de escenarios con características muy similares a las presentes en un entorno vehicular real.

Sin embargo, estos simuladores no tienen capacidad suficiente para poder simular cualquier sistema, como ocurre con el protocolo EVIGEN [Bibliografía-1] para la creación de evidencias en entornos vehiculares, siendo éste el protocolo sobre el que se va a trabajar en el presente Proyecto. Además, estos simuladores no disponen de medios suficientes para poder evaluar el rendimiento de dichos sistemas.

Aparte de esto, debido a las características presentes en las redes vehiculares, es necesario proporcionar diferentes servicios de seguridad en los protocolos ejecutados sobre las mismas. Por tanto, en este trabajo se proporcionan medios que permitan la incorporación de mecanismos de seguridad en dichos protocolos.

En definitiva, los objetivos presentes en este Proyecto son:

- Extender el simulador NCTUns 5.0 [Referencias-1] de manera que efectúe la simulación de las entidades que pertenecen al entorno de infraestructura de EVIGEN [Bibliografía-1].
- Desarrollar un sistema que permita la representación de indicadores que permitan evaluar el rendimiento de la ejecución de las entidades que participan en EVIGEN [Bibliografía-1] en un entorno concreto.
- Desarrollar un componente que ofrezca operaciones criptográficas que puedan ser utilizadas para la incorporación de servicios de seguridad en distintos protocolos simulados en NCTUns 5.0 [Referencias-1]. Entre estos servicios se debe proporcionar confidencialidad, integridad, autenticación y no repudio.

Índice de contenido

Índice de tablas

Índice de ilustraciones

1. Introducción.

Como se puede comprobar a lo largo del tiempo, las tecnologías de la información y las comunicaciones se han ido integrando y desarrollando en muchos ámbitos, creando la sociedad de la información en la que las personas habitan actualmente.

Los primeros avances destacables en este campo fueron el desarrollo de la telefonía y la televisión. Después, con la aparición de los ordenadores personales, surgió la necesidad de acceder a la información repartida por todo el mundo, por lo que se creó Internet, y en especial, la Web.

En un primer momento, únicamente se podía tener acceso a Internet a través de un modem o *router* conectado a la línea telefónica. No obstante, debido a la creciente necesidad en la sociedad de la información de obtener nuevos medios de acceso a la información proporcionada en Internet, se dedicó esfuerzo en el desarrollo de nuevas tecnologías que evitasen la necesidad de estar físicamente conectado a través de un dispositivo concreto. Entre estas tecnologías destaca Wi-Fi, encargada de proporcionar acceso a Internet a máquinas o dispositivos sin necesidad de que éstos estén conectados.

A partir de Wi-Fi, se prosiguió con el desarrollo de tecnologías para la comunicación inalámbrica en otros ámbitos, como por ejemplo en telefonía móvil o radio.

En la actualidad, otro ámbito en el que las tecnologías de la información y las comunicaciones están experimentado un gran avance, es en el desarrollo de las redes vehiculares. Dichas redes tienen como objetivo proporcionar medios para la comunicación entre vehículos, y de éstos con las infraestructuras de comunicación situadas a lo largo de las carreteras, teniendo en cuenta la fuerte heterogeneidad presente en dicho ámbito.

Por tanto, teniendo en cuenta el creciente desarrollo de las redes vehiculares, cada vez aparecen nuevas aplicaciones o servicios destinados a funcionar en dichos entornos. Estos servicios o aplicaciones presentan como objetivo principal, ofrecer mejoras en la seguridad vial o facilidades para mejorar la actividad de conducción. Por ejemplo, un servicio que se puede implantar en un entorno vehicular es la capacidad para notificar a vehículos de accidentes o atascos en distintos tramos de las carreteras, con el objetivo de que puedan variar su dirección en función de dicha información.

También, otro objetivo que se persigue conseguir con el desarrollo de las redes vehiculares es la de resolver problemas cuya solución no resulta viable en otros tipos de entornos. Entre estos problemas, uno muy común es el de no poder demostrar la no implicación en una infracción en contra de la seguridad vial, debido a que habitualmente no se dispone de medios suficientes en la actualidad para poder recabar pruebas fehacientes que permitan demostrar su inocencia a los conductores sobre dichas infracciones.

Para solucionar este problema, se propone la utilización del protocolo EVIGEN (EVIdence GENeration) [Bibliografía-1], cuyo objetivo es la creación de evidencias que permitan a un vehículo demostrar su no implicación en una infracción, siendo dichas evidencias construidas en base a los datos sobre el estado del vehículo sancionado en el momento de la infracción, procedentes de otros vehículos que se encuentran circulando próximos a éste. La descripción de este protocolo se describe en [Bibliografía-1].

No obstante, resulta necesario indicar que la implantación de una aplicación destinada a trabajar sobre entornos vehiculares presenta unos costes muy elevados, ya que para extender el servicio proporcionado por la misma a lo largo de las carreteras se necesita de la instalación de numerosas infraestructuras de comunicación. Por ello, si dicha aplicación o servicio no satisface las necesidades por las cuales ha sido creado e implantado, las pérdidas, sobre todo en el aspecto económico, serían muy elevadas.

Por este motivo, debido a las dificultades que presenta la implantación de soluciones en el ámbito vehicular, se requiere de un estudio detallado de las mismas, de forma que se pueda determinar su funcionamiento y rendimiento, con anterioridad a su instalación en dichos entornos.

Para ello, surgen los simuladores de redes vehiculares, cuyo objetivo es la simulación de entornos ficticios con características muy similares a las de los entornos vehiculares reales. Entre ellos, destaca el simulador para redes NCTUns [Referencias-1], el cual ha sido ampliamente utilizado en investigaciones anteriores. En el presente Proyecto, la versión utilizada para este simulador será la 5.0.

Sin embargo, estos simuladores no poseen la capacidad para poder especificar exactamente lo que se quiere simular en ellos, ni tampoco proporcionan medios para evaluar el rendimiento de las aplicaciones o servicios simulados en los mismos.

Por tanto, teniendo en cuenta el interés presente en implantar el protocolo EVIGEN [Bibliografía-1] en el ámbito vehicular, se debe ofrecer la posibilidad de estudiar el rendimiento y comportamiento de éste en un entorno similar al que se debería implantar en realidad. Para ello, en este Proyecto se deben implementar los medios necesarios para que se pueda realizar este estudio, los cuales son descritos en el apartado de objetivos expuesto a continuación.

Antes de concluir esta introducción, es necesario resaltar que las redes vehiculares presentan numerosas amenazas. Por ello, para contribuir a la mitigación de estas amenazas, resulta adecuado incorporar a los protocolos, determinados mecanismos criptográficos sobre los distintos mensajes intercambiados.

1.1. Objetivos del presente Proyecto.

En el presente Proyecto se pretende ampliar la funcionalidad del simulador NCTUns 5.0 [Referencias-1] con el objetivo de que pueda simular el comportamiento de las distintas entidades que participan en el protocolo EVIGEN [Bibliografía-1]. De igual modo, se pretende desarrollar un sistema de representación de indicadores que permita visualizar la información de rendimiento obtenida de la ejecución del protocolo en una simulación concreta.

Considerando que la implementación del protocolo EVIGEN [Bibliografía-1] requiere de un esfuerzo que sobrepasa el esperable en un Proyecto Fin de Carrera , este Proyecto se realiza paralelamente a otro complementario [Bibliografía-2], de modo que el protocolo se complete mediante la integración de ambos Proyectos.

Además, teniendo en cuenta las amenazas presentes en los entornos vehiculares, se debe implementar un módulo criptográfico que permita ofrecer operaciones criptográficas a distintos protocolos que puedan ser implementados y simulados en NCTUns [Referencias-1].

En la Ilustración 1 se presentan los objetivos de ambos Proyectos, la implementación del protocolo EVIGEN [Bibliografía-1] (módulo "ProtocoloEVIGEN_EntornoInfraestructura" y módulo "ProtocoloEVIGEN_EntornoInalambrico"), la creación de un sistema de representación de indicadores, un desarrollo que permita la visualización de los mensajes intercambiados en el protocolo, y un módulo que proporcione operaciones criptográficas para ofrecer servicios de seguridad a los distintos protocolos que puedan ser simulados en NCTUns [Referencias-1]. De todo ello, en el presente Proyecto se van a desarrollar los módulos coloreados de color gris, mientras que en el complementario [Bibliografía-2] se llevará a cabo el desarrollo de los módulos coloreados de morado.

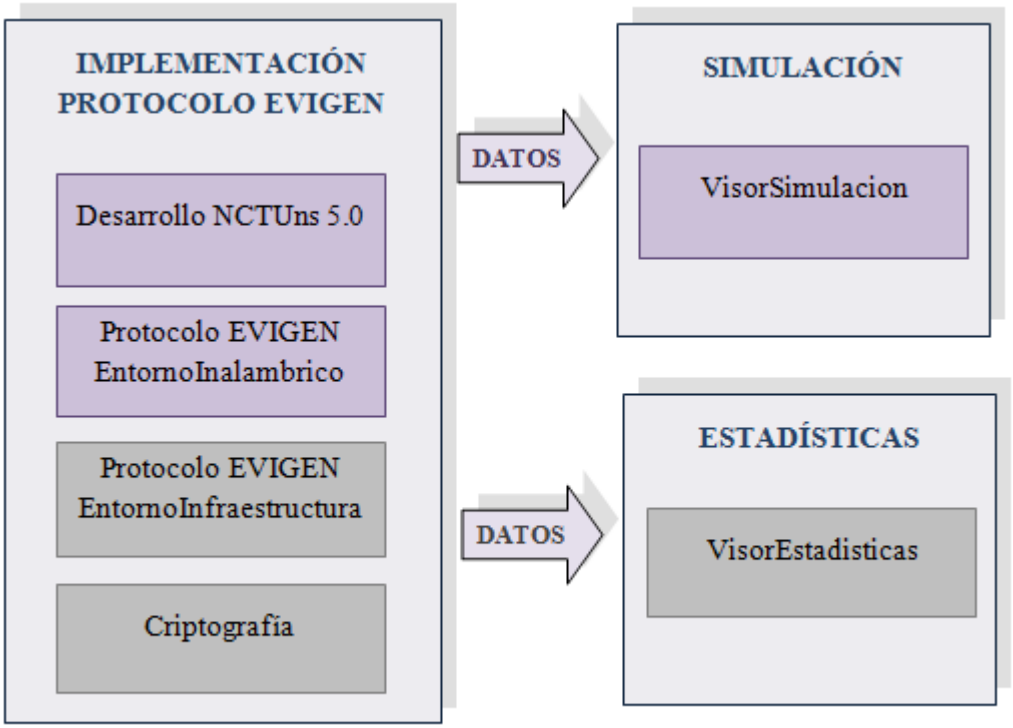

Ilustración 1: extensión simulador NCTUns.

Por un lado, el módulo "ProtocoloEVIGEN_EntornoInfraestructura" se relaciona con la implementación de las entidades que forman parte de la infraestructura de EVIGEN [Bibliografía-1].

Por otro lado, el módulo "Criptografia" se corresponde con la implementación de las operaciones criptográficas necesarias para ofrecer los servicios de seguridad en los protocolos.

Finalmente, el módulo "VisorEstadisticas" hace referencia al desarrollo del sistema encargado de la representación de los indicadores de rendimiento obtenidos de la ejecución de las distintas entidades en una simulación determinada.

En síntesis, los objetivos de este Proyecto son los siguientes:

- Extender el simulador NCTUns 5.0 [Referencias-1] de manera que efectúe la simulación de las entidades que pertenecen al entorno de infraestructura de EVIGEN [Bibliografía-1].

- Desarrollar un sistema que permita la representación de indicadores que permitan evaluar el rendimiento de la ejecución de las entidades que participan en EVIGEN [Bibliografía-1] en un entorno concreto.
- Desarrollar un componente que ofrezca operaciones criptográficas que puedan ser utilizadas para la incorporación de servicios de seguridad en distintos protocolos simulados en NCTUns 5.0 [Referencias-1]. Entre estos servicios se debe proporcionar confidencialidad, integridad, autenticación y no repudio.

1.2. Organización del presente documento.

En este apartado se describe el contenido que se va a desarrollar en cada una de las partes que componen este documento.

1. Introducción: descripción de la motivación del presente Proyecto y especificación de los objetivos que deben ser alcanzados.

2. Contexto: especificación de los fundamentos principales de las redes vehiculares, y descripción del protocolo EVIGEN [Bibliografía-1], más una serie de escenarios alternativos al mismo.

3. Análisis del sistema: especificación completa del sistema a desarrollar, de manera que queden totalmente definidos los objetivos que se pretenden conseguir con el mismo. Esta sección debe incluir principalmente un estudio de la situación actual, un estudio de las diferentes tecnologías que puedan servir de soporte para solucionar el problema, una especificación correcta y detallada de casos de uso y requisitos de software de manera que el sistema a desarrollar quede totalmente definido de forma clara y precisa, diseño arquitectónico del sistema, y diseño del plan de pruebas de aceptación.

4. Diseño detallado: descripción de la solución que debe ser desarrollada para cumplir con los requisitos expuestos en el análisis del sistema. En esta sección se presentan los diagramas con las clases que deben ser implementadas para la construcción del sistema, así como un conjunto de diagramas de secuencia que muestren la interacción lógica necesaria entre los distintos componentes para realizar la funcionalidad propia del sistema.

5. Implementación y pruebas del software: especificación de las decisiones de implementación que deban ser tomadas en dicha etapa, y realización y presentación de los resultados de las pruebas de aceptación del sistema.

6. Conclusiones y líneas futuras: exposición de las conclusiones más importantes sobre el desarrollo del Proyecto, sobre las dificultades encontradas en el mismo, y las líneas futuras.

7. Bibliografía y referencias: lista con la bibliografía consultada, así como la lista con los recursos electrónicos a los que se ha hecho referencia, durante el desarrollo del Proyecto (únicamente las referencias que aparecen en el presente documento).

1.3. Documentos complementarios.

En este apartado se indican una serie de documentos, los cuales forman también parte de la documentación de este Proyecto, y cuyo propósito es complementar el contenido presentado en esta memoria. Dichos documentos son los siguientes:

- Documento de anexos (*MemoriaPFC_Anexos.pdf*): documento independiente en el que se incluyen los anexos de esta memoria. El contenido de estos anexos es el siguiente:
  - ♦ Especificación del diseño del plan de pruebas de aceptación: especificación del diseño del plan de pruebas necesarias para la validación del sistema.
  - ♦ Evaluación del sistema: exposición de una serie de pruebas de evaluación del sistema implementado, con el fin de ilustrar la utilidad para la que ha sido creado.
  - ♦ Gestión del Proyecto: presentación de la planificación del tiempo estimado en la realización de cada una de las tareas que forman parte de este Proyecto, así como la estimación inicial del presupuesto necesario para su desarrollo. Una vez concluido este Proyecto, se debe presentar la planificación y coste real, y hacer un análisis de las desviaciones producidas.
  - ♦ Glosario de términos y acrónimos: glosario en el que se definen ciertos términos de interés para el Proyecto, y glosario en el que se especifica el significado de una serie de acrónimos que aparecen a lo largo del mismo.
  - ♦ Manual de usuario: inclusión del manual de usuario, en el que se especifiquen los pasos necesarios para poder hacer uso del software.
- Documento de diagramas de secuencia (*MemoriaPFC_DiagramaSecuencia.pdf*): documento en el que se incluyen los diagramas de secuencia que muestran la interacción concreta entre las distintas clases del sistema para desarrollar la funcionalidad propia del mismo, con el fin de servir de guía para la implementación.

2. Contexto: redes vehiculares y protocolo EVIGEN.

Especificados los objetivos del Proyecto y teniendo en cuenta que el protocolo es utilizado para simular un determinado comportamiento mediante la comunicación entre distintos vehículos y en distintos entornos, lo cual comprende una red vehicular, en esta sección se exponen los fundamentos básicos de las redes vehiculares (apartado 2.1) y la descripción del protocolo EVIGEN [Bibliografía-1] en su totalidad (apartado 2.2), incluyendo las partes "ProtocoloEVIGEN EntornoInalambrico" y "ProtocoloEVIGEN EntornoInfraestructura" presentadas en el apartado de objetivos (apartado 1.1).

Previamente al desarrollo de esta sección, es imprescindible señalar que ha sido realizada conjuntamente con el autor del proyecto complementario [Bibliografia-2].

2.1. Redes vehiculares.

Dado que el protocolo EVIGEN [Bibliografía-1] se desarrolla sobre una red vehicular, en este apartado se describen los fundamentos de este tipo de redes de comunicación.

Principalmente, en una red vehicular se distinguen dos tipos de nodos, los vehículos, nodos móviles, y las RSU (*Road Side Units*), nodos fijos. Por un lado, los vehículos constan de un pequeño hardware integrado, denominado OBU (*On Board Side Unit*), encargado de la comunicación inalámbrica con otros vehículos. Por otro lado, las RSU, dispuestas en puntos críticos a lo largo de las carreteras, se responsabilizan del envío de información hacia y entre los vehículos.

Además, en este tipo de infraestructuras también pueden existir otro tipo de nodos, como son las autoridades de certificación. Uno de los papeles fundamentales de entidades como éstas es el envío de la CRL a las distintas RSU, para que estas últimas sean las encargadas de reenviarlas a los vehículos y, de este modo, aumentar la seguridad puesto que se descartan los mensajes procedentes de un vehículo con certificado revocado.

Otro de los aspectos a considerar en las redes vehiculares es la recepción de la información del entorno, de modo que tanto vehículos como RSU dispongan de información de su entorno próximo. Esto se realiza mediante el envío de mensajes de *beaconing* a los vehículos o RSU que se encuentran situados en un determinado radio de cercanía, lo cual queda reflejado en la Ilustración 2.

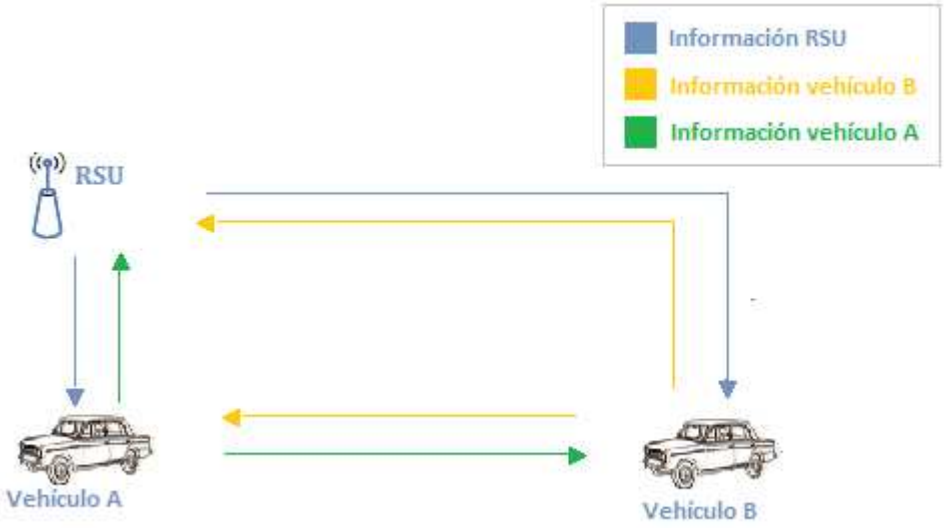

Ilustración 2: envío mensaje de *beaconing*.

2.2. Descripción del protocolo EVIGEN implementado en este proyecto.

El propósito del protocolo EVIGEN [Bibliografía-1] (cuyo nombre procede de EVIdence GENeration) es el de de permitir la creación de pruebas electrónicas en el contexto del procedimiento sancionador que permitan describir el comportamiento del vehículo en el momento al que se refiere la sanción. Para conseguir este objetivo, la aproximación de EVIGEN [Bibliografía-1] se basa en la comunicación del vehículo sancionado con otros cercanos, los cuales puedan emitir una descripción ("testimonio") de dicho comportamiento. Como ejemplo de aplicación, este protocolo se podría ejecutar si un vehículo recibiera una notificación de sanción por exceso de velocidad, el cual es el enfoque en el que se basa la descripción del protocolo y el que sustenta el desarrollo de este Proyecto.

2.2.1. Roles participantes.

Teniendo en cuenta los distintos nodos que pueden existir en una red vehicular (introducidos en el apartado 2.1), en EVIGEN [Bibliografía-1] se identifican distintos roles para cada uno de ellos. En particular los nodos móviles pueden tomar tres roles distintos: vehículo infractor (*Requester*), vehículo testigo (*Witness)* y vehículo no equipado (*NoEquipped*), mientras que los nodos fijos se corresponden con RSU, AC y DGT. Es conveniente indicar que un vehículo, en un primer momento, sólo puede ser testigo o no equipado, pasando, conjuntamente, a tomar el rol de vehículo infractor en el momento de recepción de un mensaje procedente de una RSU, en este caso correspondiente con una notificación de sanción.

Previamente a la descripción del protocolo y con el propósito de facilitar la comprensión del mismo, se presentan las funciones asociadas a cada uno de los roles:

- Vehículo infractor (*Requester*): encargado de solicitar a los vehículos testigos, que se encuentren dentro de su radio de cercanía, sobre la velocidad a la que iba en un determinado momento. Tras recibir y procesar los testimonios recibidos, ha de construir la evidencia y, en caso de que la velocidad resultante sea inferior a la que fue sancionado, debe enviar dicha

evidencia a la RSU más cercana para que posteriormente sea recibida en la DGT.

- Vehículo testigo (*Witness*): encargado de aportar los testimonios de acuerdo con una solicitud realizada por un vehículo infractor. El testimonio creado puede ser enviado al propio vehículo infractor o, en caso de no encontrarse éste lo suficientemente cerca, se enviaría a las entidades de tipo vehículo no equipado y a las RSU cercanas al testigo. Teóricamente, se corresponde con una entidad con sensores capaces de detectar los acontecimientos ocurridos en un determinado radio.
- Vehículo no equipado (*NoEquipped*): encargado de realizar reenvíos de testimonios para que el vehículo infractor pertinente pueda recibirlo. En caso de no poder enviarlo al vehículo infractor, por no encontrarse cerca, se enviaría al resto de vehículos no equipados y RSU próximas al vehículo no equipado. Teóricamente, representa a aquellos vehículos que no disponen de información sensorial útil para el desarrollo del protocolo pero que, sin embargo, están equipados con una unidad de comunicaciones a bordo (OBU).
- RSU: encargada de la detección de infracciones cometidas por vehículos, y de su notificación a la DGT. Además, actúan de intermediarios de los mensajes que transmiten los distintos participantes del protocolo. Teóricamente representan a las unidades que se encuentran bajo el control del gobierno, situadas a lo largo de las carreteras.
- DGT: encargada de la emisión de las sanciones de los vehículos, de la verificación de las evidencias que demuestran la inocencia de los mismos, y de determinar los vehículos cuyo certificado se debe revocar. Teóricamente representa una autoridad legal.
- AC: encargada de la gestión de la CRL y de su transmisión a las RSU. Teóricamente asume el rol de la autoridad de certificación emisora de los certificados de cada una de las entidades del protocolo. Esta entidad no participa de forma directa en el protocolo, pero el soporte que ofrece para la revocación de los certificados de los vehículos infractores, implica cambios en el comportamiento del mismo.

2.2.2. Descripción básica de los mensajes intercambiados.

Con el propósito de clarificar el intercambio de mensajes producido, la descripción del protocolo se va a realizar mediante la creación del entorno más simple, el cual se presenta posteriormente, en la Ilustración 3.

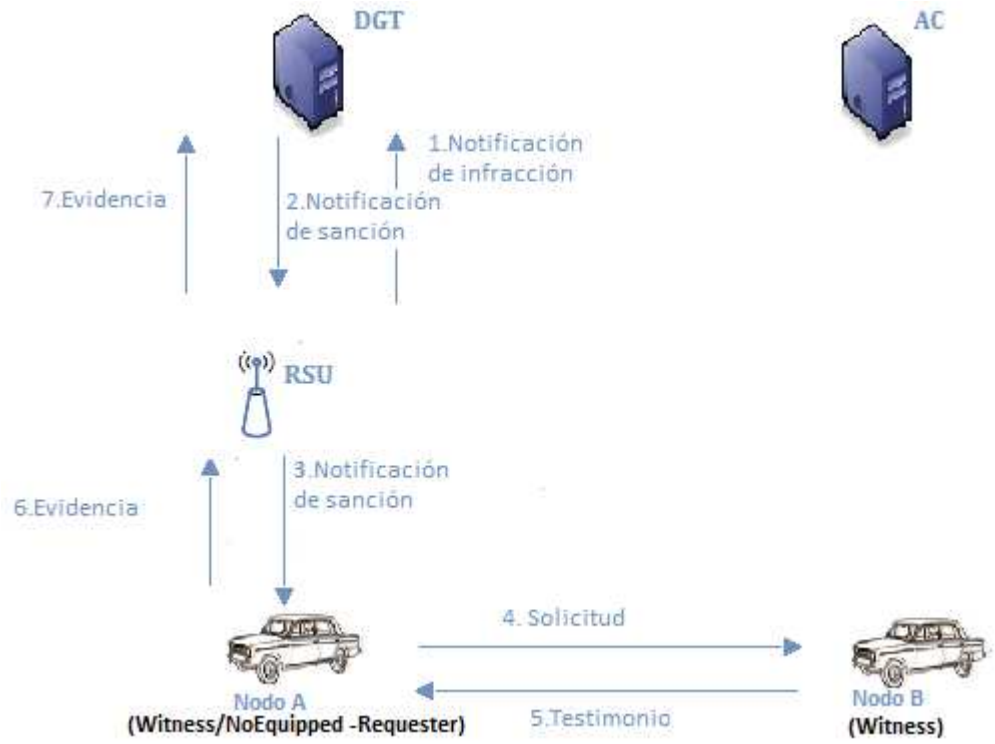

Ilustración 3: descripción básica del protocolo EVIGEN.

La descripción, de acuerdo con los mensajes presentado en la Ilustración 3, es la siguiente:

1. La RSU detecta que un vehículo, en adelante nodo A, circula por encima del límite de velocidad establecido, notificándoselo a la DGT. Este mensaje de notificación de infracción a la DGT no entra dentro del ámbito del protocolo, sino que su objetivo es servir de motivación a la DGT para iniciar la ejecución del mismo.

2. La DGT genera la notificación correspondiente y se la envía a la RSU para que ésta realice el envío al nodo oportuno. Este es el primer mensaje enviado del protocolo.

3. La RSU envía la notificación de la sanción al nodo A.

4. El nodo A, el cual asume el rol de vehículo infractor, envía una solicitud a un vehículo testigo, en adelante nodo B, para obtener un testimonio y poder revocar la sanción.

5. El nodo B, vehículo testigo, responde al nodo A con un testimonio. (En caso de haberse desplazado lo suficientemente lejos como para que el mensaje no pudiese ser recibido por el nodo A, se debería realizar un reenvío a los vehículos no equipados o RSU que se considerasen cercanos).

6. El nodo A crea la evidencia asociada al testimonio recibido y la envía a la RSU.

7. La RSU realiza el envío de la evidencia a la DGT.

2.2.3. Escenario alternativo.

Conocido el funcionamiento principal del protocolo, se presenta un escenario que permanece al margen del mismo, pero cuyo conocimiento es necesario para la correcta comprensión del desarrollo presentado.

Este escenario se corresponde con lo que sucede si la DGT no recibe la evidencia asociada a una notificación de sanción enviada o si el dato de consenso de dicha evidencia no se corresponde con los testimonios de los testigos, lo cual puede ocurrir por distintos motivos. Por una parte, si alguno de los mensajes intercambiados en el protocolo se pierde (considerar el envío inalámbrico), la evidencia no podría ser enviada. Otra posible circunstancia es que un vehículo no envíe una evidencia no favorable aunque el protocolo se desarrollase adecuadamente, lo cual se asume. Finalmente, también puede suceder que un vehículo construya una evidencia cuyo dato no se corresponde con los testimonios aportados por los testigos, enviándola a la DGT para poder evitar la sanción.

La descripción de este escenario se realiza bajo la suposición de que se produce un fallo en la recepción de uno de los mensajes del protocolo y, por ello, la evidencia no puede ser enviada a la DGT. El desarrollo se muestra posteriormente, en la Ilustración 4.

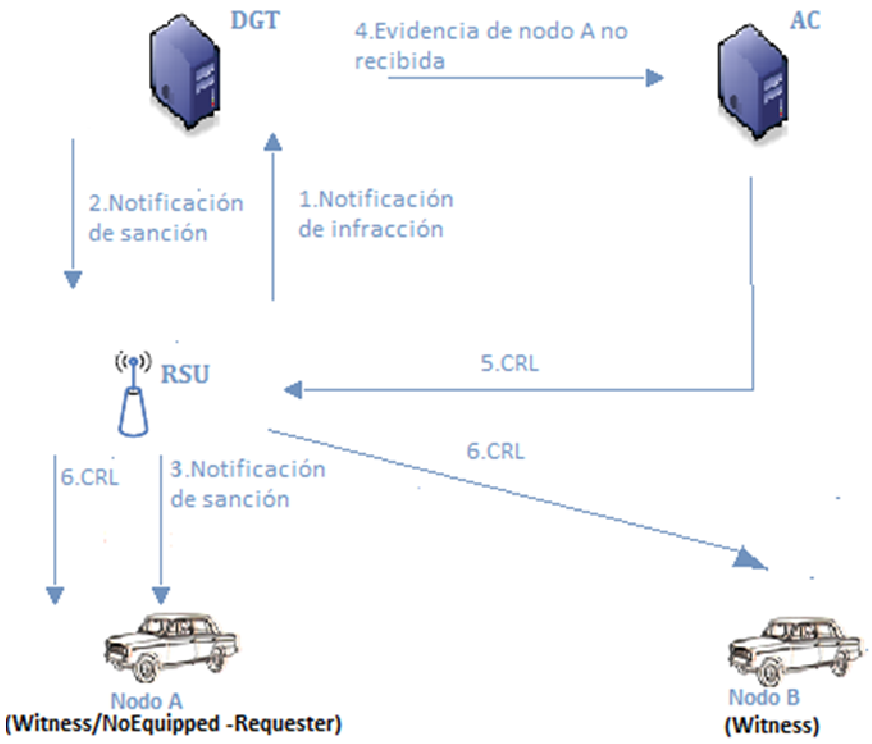

Ilustración 4: escenario alternativo vinculado al desarrollo del protocolo EVIGEN.

La descripción es la siguiente:

1. La RSU detecta que un vehículo, en adelante nodo A, circula por encima del límite de velocidad establecido, notificándoselo a la DGT.

2. La DGT genera la notificación correspondiente y se la envía a la RSU para que ésta realice el envío al nodo oportuno.

3. La RSU envía la notificación de la sanción al nodo A.

4. Pasado un tiempo, la DGT no recibe respuesta y envía a la AC un mensaje con el identificador del nodo A para revocar su certificado

5. La AC revoca el certificado del nodo A, introduciéndolo en la CRL y enviándosela a la RSU para propagar la modificación realizada.

6. La RSU envía, a todos los nodos que permanezcan en su radio de cercanía, la CRL recibida.

3. Análisis.

En esta sección, se realiza el análisis del sistema que se pretende desarrollar. En los párrafos siguientes, se describe el contenido que compone esta sección.

En primer lugar, se hace un estudio de la situación actual. En concreto, se hace un estudio del simulador NCTUns 5.0 [Referencias-1] para determinar la forma en la que debe ampliarse su funcionalidad para la simulación del entorno de infraestructura de EVIGEN [Bibliografía-1], y para representar la información del rendimiento de las entidades.

Posteriormente, se hace una descripción del sistema que se debe desarrollar para cumplir con los objetivos definidos en el apartado 1.1.

Después, se presenta un diagrama de componentes que refleje la funcionalidad descrita en la descripción del sistema, y se especifican las tecnologías impuestas por el entorno en el que se ubicará el mismo.

También se incluye un análisis de las posibles amenazas y vulnerabilidades que se pueden presentar en el sistema, indicando el riesgo que éstas suponen y las posibles soluciones que se pueden aplicar para su prevención.

Además, se hace un estudio tecnológico con el objetivo de determinar las diferentes tecnologías existentes que se pueden utilizar para el desarrollo del sistema.

En esta sección, se presenta también la arquitectura de alto nivel del sistema, y se indican las tecnologías que han sido finalmente seleccionadas del estudio realizado al respecto.

También se presenta el diagrama de casos de uso, y se hace una descripción de cada uno de ellos.

En esta sección se incluye también el contenido más importante del análisis, la especificación de los requisitos de software del sistema que se va a desarrollar.

Finalmente, se realiza la especificación del diseño de las pruebas que se van a efectuar para poder validar la funcionalidad del sistema.

3.1. Estudio de la situación actual.

Como se ha especificado en la introducción, teniendo en cuenta que el protocolo EVIGEN [Bibliografía-1] está pensado para su uso en entornos vehiculares, y que su rendimiento debe ser evaluado en dicho ámbito, se necesita un medio que consiga simularlo.

Para ello, se ha decidido utilizar el simulador NCTUns 5.0 [Referencias-1], el cual presenta grandes ventajas para la creación de un entorno vehicular simulado con prestaciones muy similares a las de un entorno real, además de proporcionar facilidades para la comunicación entre los diferentes nodos que puedan formar parte de una simulación, haciendo uso de distintos protocolos de red (por ejemplo, 802.11p para la comunicación en redes vehiculares).

Por este motivo, con el fin de poder integrar este protocolo en NCTUns 5.0 [Referencias-1], de manera que se pueda simular el mismo, es necesario realizar un estudio previo del funcionamiento interno del simulador para poder determinar cómo realiza realmente las simulaciones, y de esta manera, establecer las medidas necesarias que se deben tener en cuenta para su integración.

3.1.1. Estudio del simulador NCTUns 5.0.

Del estudio realizado, en este apartado únicamente se van a incluir los aspectos principales del simulador que deben ser considerados para el desarrollo de las entidades del entorno de infraestructura del protocolo EVIGEN [Bibliografía-1] que deben ser simuladas en el mismo.

En primer lugar, cuando en el simulador, desde la interfaz gráfica (GUI), se crea un entorno con una serie de nodos, a cada uno de ellos se le debe asociar un rol que será el que defina su comportamiento durante la simulación.

Un rol se trata de un programa independientemente, cuyo objetivo debe ser la especificación del comportamiento del nodo al que es asociado. Por ejemplo, si el rol de la entidad DGT consiste, entre otras cosas, de crear sanciones a los vehículos infractores, el programa creado debe implementar dicha creación de sanciones. Además, en caso de que un nodo deba comunicarse con el resto, en el programa que define su rol se deben implementar todos los mecanismos que sean necesarios para que se establezca dicha comunicación.

De esta manera, cuando se tiene el entorno simulado creado con una serie de nodos, y cada uno de ellos tiene especificado su rol, se ejecuta la simulación, y el simulador por sí mismo, crea un proceso en el que se realizan todas las operaciones necesarias para el desarrollo de la simulación. Entre estas operaciones, se encarga de la creación, por cada nodo definido en el entorno simulado, de un proceso independiente (proceso hijo del proceso encargado de realizar la simulación) en la máquina local en la que se realiza la simulación, en el cual se ejecuta el programa correspondiente con el rol definido para ese nodo.

Por este motivo, para el desarrollo del protocolo EVIGEN [Bibliografía-1], por cada entidad que participe en el mismo, se debe crear un programa independiente que defina su comportamiento, y luego especificar el nombre del mismo como rol de un nodo concreto en el entorno a simular, para que sea ejecutado por el simulador cuando se inicie la simulación.

Además, con el fin de facilitar la comunicación entre los procesos, en la ejecución de la simulación, NCTUns [Referencias-1] asigna a cada uno de ellos una dirección IP propia.

Otro aspecto importante que debe ser tenido en consideración para el desarrollo de las entidades del protocolo, es que en la ejecución de la simulación, el simulador crea un directorio concreto en el que se almacenan todos los ficheros de configuración necesarios para la simulación. Entre estos ficheros, se encuentra uno con información propia de los nodos, como por ejemplo, posición inicial en la que se encuentran, velocidad inicial, etc.

Esta información propia de los nodos es necesaria para la correcta ejecución de los mismos. Por ello, los programas que ejecutan el rol de éstos pueden acceder al contenido de los ficheros de este directorio, debido a que la ruta del mismo es especificada por el proceso que ejecuta la simulación (proceso padre de los procesos que ejecutan los roles de los nodos), como variable de entorno.

También puede obtenerse información directamente del kernel utilizado por NCTUns [Referencias-1] para la realización de las simulaciones, como por ejemplo, el identificador único asociado a cada uno de los nodos.

Finalmente, se describen una serie de restricciones encontradas en el simulador durante su estudio, las cuales van a afectar en la implementación del sistema.

La primera de ellas es que no se dispone del código fuente de la GUI, por lo que no se puede modificar la misma para añadir más funcionalidad, como por ejemplo, insertar parámetros de configuración a las distintas entidades. Además, debido a esta restricción, no se puede integrar en NCTUns [Referencias-1] como ampliación del mismo, el sistema que permita la representación gráfica de los indicadores de rendimiento del protocolo. Por tanto, este sistema de representación de indicadores debe ser desarrollado de manera independiente al simulador.

La otra restricción consiste en que, si en el entorno a crear se disponen de varias entidades conectadas físicamente entre sí, se comprobaría que, por cada par de nodos comunicados, NCTUns [Referencias-1] crea una red distinta. Esto supone un inconveniente ya que si una entidad concreta pertenece a varias redes diferentes, el simulador le asigna una dirección IP diferente por cada una de ellas. De esta forma, la entidad no puede tener conocimiento de las diferentes direcciones por las que se debe comunicar con el resto, ya que de forma autónoma sólo puede conocer una, teniendo que ser especificadas de manera explícita por el usuario, antes de comenzar cada simulación. Por ello, con el objetivo de que los nodos puedan determinar su dirección por sí solos en tiempo de ejecución, deben pertenecer a la misma red. Además, otro motivo por el que todas las entidades deben situarse en la misma red, es que si se hace *broadcast* para enviar un mensaje a todos los nodos, éste solamente funciona si se realiza en la dirección "1.0.1.255", por lo que todos los nodos deben pertenecer a la red "1.0.1.X".

3.2. Descripción del sistema.

En este apartado se describe el sistema que debe ser desarrollado para el cumplimiento de los objetivos expuestos en el apartado 1.1.

Teniendo en cuenta el estudio del simulador NCTUns [Referencias-1] efectuado en el apartado 3.1, se comprueba que para la simulación de las entidades de la infraestructura de EVIGEN [Bibliografía-1] se deben crear los programas correspondientes con cada entidad de manera independiente al simulador, ya que al ejecutar una simulación concreta, éste se encarga de ejecutar los programas correspondientes con las entidades que participan en dicho escenario. Además, teniendo en cuenta que no se dispone del código fuente de la GUI del simulador, no se puede ampliar la misma, por lo que el sistema encargado de representar la información de rendimiento debe ser implementado también de manera independiente.

Por tanto, considerando lo comentado en el párrafo anterior, en el sistema se pueden distinguir dos partes que pueden ser desarrolladas de manera independiente:

- Extensión de NCTUns [Referencias-1] para la simulación del entorno de infraestructura del protocolo EVIGEN [Bibliografía-1].
- Desarrollo del sistema que permita la representación, de manera gráfica, de los indicadores de rendimiento obtenidos de la ejecución de las distintas entidades.

En lo que respecta a la creación del módulo criptográfico, teniendo en cuenta que el interés de su uso se encuentra en proporcionar seguridad a los distintos protocolos que puedan ser simulados, éste va a ser diseñado e implementado como un componente independiente, vinculado a la parte del sistema dedicado a la extensión de NCTUns [Referencias-1] para la simulación de las entidades de la infraestructura del protocolo (ver apartado 3.3.1.1).

Por este motivo, especificadas las dos partes distintas en las que se distingue el sistema, con el fin de facilitar la explicación, se va a describir cada una de estas partes, por separado, en los dos subapartados siguientes.

3.2.1. Descripción de la extensión de NCTUns para la simulación del entorno de infraestructura de EVIGEN.

Teniendo en cuenta que las entidades que componen la infraestructura de EVIGEN [Bibliografía-1] son la AC, DGT y RSU, y considerando la descripción del protocolo efectuada en el apartado 2.2, se va a describir en mayor profundidad la funcionalidad asociada a estas entidades, centrándose especialmente en el contenido de los distintos mensajes que transmiten y otros aspectos no especificados en el apartado de descripción del protocolo, dividiendo dicha descripción según distintos aspectos para una mayor claridad.

3.2.1.1. Escenario básico del entorno de infraestructura de EVIGEN.

Como se puede comprobar en el apartado 2.2, cuando una RSU detecta una infracción, crea una notificación con la información de la misma y se lo transmite en un mensaje a la DGT. A continuación, se muestra la estructura de esta notificación:

IdVehiculo, LocVehiculo, VelVehiculo, IdInfraccion, IdRSU , HoraInfraccion

- *IdVehiculo* : identificador del vehículo infractor.
- *LocVehiculo* : localización del vehículo en el instante en el que se cometió la infracción.
- *VelVehiculo* : velocidad del vehículo en el momento en el que se detectó la infracción.
- *IdInfraccion* : identificador de la infracción que ha cometido el vehículo.
- *IdRSU* : identificador de la RSU que ha detectado la infracción.
- *HoraInfraccion* : fecha y hora en la que se produjo la infracción.

Teniendo en cuenta que el número de sanciones en seguridad vial es elevado, y que la importancia del protocolo no radica en este punto, en el sistema a crear el tipo de la infracción se va a limitar a exceso de velocidad.

Resulta necesario indicar que la RSU únicamente puede realizar el envío a la DGT si se encuentra directamente conectada con ella. En caso contrario, la RSU debe transmitir el mensaje a sus RSU vecinas, y así sucesivamente, hasta que una de ellas pueda enviarlo a la DGT.

Además, con el fin de no enviar numerosas notificaciones de infracción a la DGT, una RSU únicamente puede enviar una notificación de infracción de un vehículo, si no ha enviado otra notificación del mismo vehículo en un intervalo de tiempo determinado.

Una vez que la DGT recibe la notificación de la infracción de un vehículo, a partir de la información contenida en ella, crea una notificación de sanción para comunicarle al vehículo infractor de la sanción que ha cometido, y se lo envía en un mensaje a las RSU directamente conectadas a ella, con el objetivo de que lo vayan retransmitiendo entre ellas hasta llegar al vehículo infractor. El formato de la notificación de sanción es el siguiente:

IdVehiculo, IdInfraccion, IdNotificacion, IdRSU , HoraInfraccion, LocVehiculo, DatoInfraccion

- *IdVehiculo* : identificador del vehículo que ha infringido una norma de seguridad vial.
- *IdInfraccion* : identificador de la infracción que ha cometido el vehículo infractor.
- *IdNotificacion* : identificador único asociado a la notificación de sanción.
- *IdRSU* : identificador de la RSU que ha detectado la infracción del vehículo.
- *HoraInfraccion* : fecha y hora del momento en el que fue detectada la correspondiente infracción.
- *LocVehiculo* : localización del vehículo en el instante en el que la RSU detectó la infracción cometida por éste.

- $DatoInfraccion$: información concreta al tipo de infracción cometida en la cual se comunica al vehículo infractor los datos necesarios que demuestran que ha infringido una norma de seguridad vial. Teniendo en cuenta que en el sistema presente el tipo de infracción se limita a exceso de velocidad, esta información consiste en indicar la velocidad a la que circulaba el vehículo cuando la RSU correspondiente detectó la infracción.

Es necesario resaltar que, en caso de que la DGT reciba múltiples notificaciones de infracción de un mismo vehículo, procedentes de distintas RSU en un intervalo de tiempo determinado, únicamente se debe enviar una notificación de sanción correspondiente con la primera notificación de infracción obtenida para ese vehículo. El objetivo de esto es evitar que la DGT mande varias notificaciones de sanción a un mismo vehículo, antes de que éste pueda responder a la primera de ellas.

Una vez que la notificación de sanción haya sido transmitida al vehículo infractor, éste se debe encargar de recoger los testimonios de los posibles testigos de la infracción y construir una evidencia cuyo objetivo sea el de recurrir la correspondiente sanción.

Construida la evidencia de respuesta a la notificación de sanción, el vehículo infractor la transmite dentro de un mensaje a la RSU que se encuentra más próxima a la ubicación por la que se encuentra circulando en ese instante. Una vez recibido en esa RSU, ésta se encarga de reenviarlo a la DGT en caso de encontrarse conectado directamente a ella. En caso de no ser así, se envía a las RSU cercanas y así sucesivamente, hasta que el mensaje se reciba en la DGT. A continuación, se presenta el formato de la evidencia:

$$IdNotificacion, DatoConsenso, (Ticket, CondTestimonio)^{+}$$

$(contenido)^{+}$ = el valor de *contenido* debe repetirse de 1 a infinitas veces.

- $IdNotificacion$: identificador de la notificación de sanción a la cual responde la correspondiente evidencia.
- $DatoConsenso$: dato construido en función de los testimonios aportados por los distintos testigos, con el fin de obtener la prueba definitiva que demuestre que el vehículo no ha cometido la infracción que le ha sido imputada. En la infracción por exceso de velocidad, el dato de consenso es la velocidad a la que circulaba el vehículo en el instante de la infracción, obtenida a partir de los testimonios de los testigos.
- $Ticket$: información dirigida a la DGT para autenticar al testigo y garantizar el no repudio de las condiciones del testimonio asociadas a este ticket. Este ticket presenta la siguiente forma:

$$E_{Kpub(DGT)}(Cert(Testigo), S_{Kpriv(Testigo)}(KTestimonio(Testigo)))$$

  - $Cert(Testigo)$: certificado digital del testigo que emite el correspondiente testimonio.

- $S_{Kpriv(Tetigo)}(KTestimonio(Testigo))$: clave simétrica con la que han sido cifradas las condiciones del testimonio, y firma digital de la misma.
- $KTestimonio(Testigo)$: clave simétrica que se ha utilizado para el cifrado de las condiciones del testimonio.

Se puede comprobar que el ticket está cifrado con la clave pública de la DGT, impidiendo, por tanto, que esta información pueda ser observada por otras entidades.

- *CondTestimonio*: información asociada a la evidencia cuyo fin es el de garantizar, a la DGT, que el dato de consenso ha sido generado en función de los testimonios aportados por los testigos. Teniendo en cuenta que en el sistema a desarrollar el tipo de la infracción es por exceso de velocidad, la información contenida en las condiciones del testimonio es un intervalo en el que se sitúa la velocidad del vehículo en el momento de la infracción, de acuerdo a la información aportada por el testigo correspondiente. A continuación, se muestra la estructura de las condiciones del testimonio, para el caso de exceso de velocidad:

$$SE_{KTestimonio(Testigo)}(\text{int } ervalo\_\min, \text{int } ervalo\_\max)$$

  - $Intervalo\_\min$: extremo mínimo del intervalo en el que se ubica la velocidad del vehículo infractor.
  - $Intervalo\_\max$: extremo máximo del intervalo en el que se encuentra la velocidad del vehículo infractor.

  Como se puede observar, las condiciones del testimonio se encuentran cifradas con la clave simétrica contenida en el ticket, consiguiendo de esta manera confidencialidad, siendo la DGT la única que puede obtener su contenido (ya que el ticket que contiene la clave sólo puede ser descifrado por la DGT), y además, se asegura el no repudio de su contenido, debido a que la clave simétrica contenida en el ticket se encuentra firmada por el testigo que la ha generado.

Una vez recibida la evidencia en la DGT, ésta debe comprobar que el dato de consenso alcanzado se encuentra avalado por las condiciones de los testimonios incluidos en la evidencia.

3.2.1.2. Escenario alternativo de EVIGEN.

Como se contempló en el escenario alternativo del protocolo expuesto en el apartado 2.2.3, en caso de que la DGT reciba una evidencia con un dato de consenso que no se corresponda con los testimonios de los testigos, o no se obtenga respuesta a una notificación de sanción, se envía un mensaje a la AC con el identificador del correspondiente vehículo infractor para revocar su certificado. Es necesario indicar que en el caso de no recibir respuesta a una notificación de sanción en un tiempo determinado, si se dispone de un número máximo de reenvíos, la DGT debe hacer primeramente el reenvío de dicho mensaje hasta dicho número máximo de veces.

Cuando la AC recibe el mensaje con el identificador del vehículo cuyo certificado se desea revocar, lo incluye en la CRL, transmitiéndola tras su

actualización a las RSU conectadas a ella. Una vez recibida en las RSU, éstas se encargan de ir propagando el mensaje con la CRL al resto de RSU conectadas con ellas y a los vehículos que se encuentren circulando próximas a ellas.

Es necesario indicar que, con el objetivo de asegurar que todos los vehículos que se encuentran circulando tengan conocimiento de los vehículos cuyo certificado ha sido revocado, la AC transmite de manera periódica la CRL cada intervalo de tiempo determinado.

Por otro lado, las RSU también tienen como función tomar el rol de vehículo de tipo no equipado, los cuales tienen como objetivo actuar de intermediarios en el envío de los testimonios creados por los distintos testigos al vehículo infractor.

Por este motivo, una RSU puede recibir un mensaje con un testimonio procedente del testigo, de otra RSU o de un vehículo no equipado, y debe reenviarlo al vehículo infractor en caso de que se encuentre circulando próxima a ella, o al resto de RSU vecinas y vehículos de tipo de no equipado cercanos a ella por una distancia máxima de comunicación. El formato del mensaje con el testimonio es el siguiente:

$$IdVehiculoInfractor\,,\,E_{Kpub(VehiculoInfractor)}(Testimonio)$$

- *IdVehiculoInfractor* : identificador del vehículo infractor al que va dirigido el testimonio.
- *Testimonio* : contenido del testimonio. Esta información se encuentra cifrada, de manera que únicamente el vehículo infractor pueda acceder a su contenido.

3.2.1.3. *Beacon* de RSU.

Resulta imprescindible comentar que, con el fin de que los vehículos puedan conocer las RSU que se encuentran situadas a lo largo de las carreteras para comunicarse con ellas, éstas envían cada cierto periodo de tiempo un *beacon* a los vehículos que se encuentran circulando próximas a ellas informando del estado de las mismas. La estructura de este *beacon* es la que se presenta a continuación:

$$IdRSU\,,\,LocRSU$$

- *IdRSU* : identificador de la RSU que emite el *beacon*.
- *LocRSU* : localización de la RSU.

3.2.1.4. Log de operaciones criptográficas.

Además, teniendo en cuenta que las distintas entidades realizan una serie de operaciones criptográficas durante su ejecución, por cada operación realizada, se debe almacenar en un log una entrada que informe de la misma.

3.2.1.5. Sistema de visualización de mensajes.

Por otro lado, aunque el presente Proyecto no abarca la realización del sistema de visualización de los mensajes intercambiados por las entidades (su creación forma parte del Proyecto complementario a éste [Bibliografia-2]), es necesario enviar a dicho sistema información relativa al intercambio de información de los mensajes del protocolo entre las RSU y los vehículos, así como el contenido de los *beacon*.

3.2.2. Descripción del sistema de representación de indicadores.

En este subapartado, se efectúa la descripción del sistema encargado de la representación de los indicadores.

Durante la ejecución del protocolo, cada una de las distintas entidades que participan en él debe obtener una serie de indicadores que permitan evaluar el rendimiento del mismo. El tipo de estos indicadores se especifica en los puntos siguientes:

- Tamaño de los mensajes: número de bytes transmitidos en los diferentes mensajes intercambiados por las entidades.
- Tiempo de envío: cantidad de milisegundos empleados en el envío de los mensajes transmitidos por las entidades.
- Tiempo de respuesta: número de milisegundos que se tarda en recibir los mensajes con los distintos testimonios aportados por los testigos en los vehículos infractores que realizan la correspondiente solicitud, y número de milisegundos empleados en la recepción de los mensajes con evidencias de respuesta a determinadas notificaciones de sanción en la DGT.
- Tiempo de computación: cantidad de milisegundos necesarios para la realización de las operaciones criptográficas del protocolo.
- Éxito o fracaso en la finalización del protocolo: dato que indique si el protocolo se ejecuta correctamente o incorrectamente. El protocolo se ejecuta correctamente si la DGT recibe una evidencia de respuesta a una notificación de sanción, y fracasa si transcurre el tiempo de espera máximo de recepción de la evidencia y ésta no se ha recibido.
- Aceptación o rechazo en la participación en el protocolo: dato que señala si un determinado vehículo acepta la participación en el protocolo, o rechaza dicha participación porque se encuentra computacionalmente saturado.

Cada vez que una entidad del protocolo obtenga un indicador, debe enviar el valor del mismo al sistema de representación de indicadores. Este sistema se debe encargar de reflejar gráficamente, haciendo uso de dichos indicadores, el rendimiento del protocolo sobre el que se obtienen los mismos.

Este sistema debe representar la información obtenida de los indicadores de rendimiento de dos maneras distintas.

La primera de ellas debe mostrar la información básica de los indicadores de rendimiento, la cual se compone del número total de indicadores obtenidos de cada tipo; medias del tamaño de los mensajes, tiempo de envío, tiempo de respuesta y tiempo de computación; tasas de éxito y fracaso en la finalización del protocolo, y tasas de aceptación y rechazo del mismo.

La segunda debe presentar la información básica expuesta en el párrafo anterior, además de gráficas que muestren el valor de cada uno de los indicadores obtenidos del tamaño de los mensajes, tiempo de envío, tiempo de respuesta y tiempo de computación; y las tasas de éxito y fracaso, y aceptación y rechazo.

Además, desde el sistema se debe permitir al usuario realizar las operaciones necesarias para su gestión:

- Comenzar el sistema: comenzar la representación de indicadores de rendimiento. Previamente a ello, el sistema debe eliminar todas las representaciones de indicadores anteriores que pudiese contener.
- Parar el sistema: parar la representación de los indicadores que se está realizando en un momento concreto. Una vez parado el sistema, se debe comenzar de nuevo el mismo para la obtención de nuevos indicadores.
- Pausar el sistema: parar la presentación de indicadores que se está efectuando en un instante determinado. No obstante, a diferencia de la operación de parada, se puede continuar con la representación sin necesidad de reiniciar el sistema.
- Continuar el sistema: continuar la representación de indicadores de rendimiento, sin necesidad de tener que reiniciar el sistema previamente.
- Almacenar estadísticas en formato PDF: se deben reflejar las estadísticas mostradas por la interfaz gráfica, en modo avanzado, en formato PDF.
- Almacenar valor de los indicadores de rendimiento en formato Excel 2003: almacenar el valor de los indicadores obtenidos en las distintas entidades del protocolo, en formato Excel 2003.
- Cargar valor de los indicadores de rendimiento en formato Excel 2003: obtener, de un fichero en formato Excel 2003, el valor de los indicadores de rendimiento que se encuentran almacenados en éste, y representarlos en el sistema.

3.3. Arquitectura preliminar y análisis de las tecnologías impuestas.

En este apartado se presenta la arquitectura preliminar del sistema, y se seleccionan las tecnologías impuestas por el entorno. Se subdivide en dos subapartados distintos:

- Arquitectura preliminar: diagrama, de muy alto nivel, que presenta los componentes necesarios para construir el sistema.
- Análisis de la tecnologías impuestas: especificación de las tecnologías impuestas para el desarrollo del sistema, justificando las razones por las que se hace dicha elección.

3.3.1. Arquitectura preliminar.

En este subapartado se presenta la arquitectura preliminar del sistema.

Teniendo en cuenta las dos partes distintas descritas en el apartado 3.2, correspondientes con la implementación de las entidades pertenecientes a la infraestructura del protocolo EVIGEN [Bibliografía-1] y el sistema de representación de indicadores, en los subapartados siguientes se presentan dos diagramas de componentes diferentes, cada uno de acuerdo a cada una de estas partes.

Además, previamente a la presentación de la arquitectura de cada parte del sistema, en la Ilustración 5 se presenta el flujo de información existente entre la implementación de las entidades de EVIGEN ("Implementación Protocolo EVIGEN") (tanto del entorno de infraestructura, como del entorno inalámbrico desarrollado en el Proyecto complementario [Bibliografía-2]) y el sistema de

representación de indicadores ("VisorEstadísticas"), ya que este último debe representar gráficamente la información de los indicadores de rendimiento obtenidos de la ejecución de las entidades.

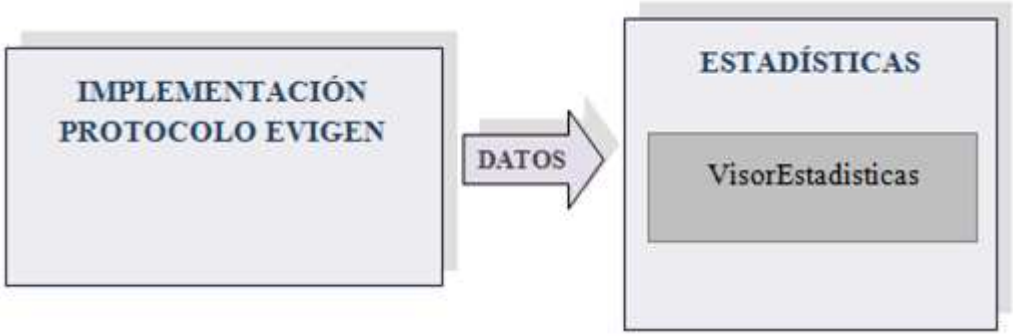

Ilustración 5: flujo de datos.

3.3.1.1. Arquitectura preliminar de la extensión de NCTUns para la simulación del entorno de infraestructura del protocolo EVIGEN.

En la Ilustración 6 se muestra el diagrama de componentes correspondiente con la extensión del simulador para la simulación de las entidades correspondientes con la infraestructura de EVIGEN [Bibliografía-1].

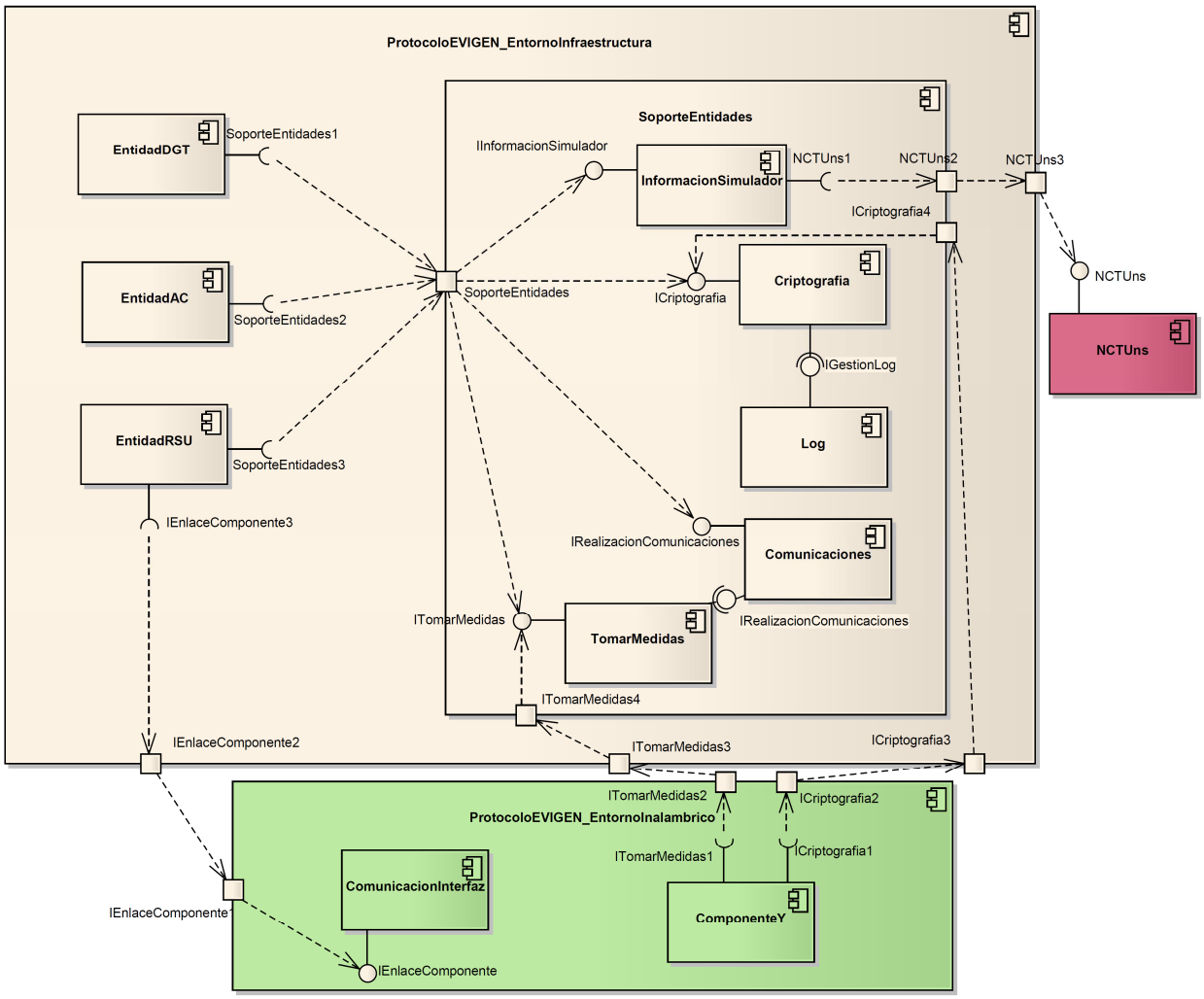

Ilustración 6: arquitectura preliminar de extensión NCTUns para simulación de entidades de infraestructura de EVIGEN.

En la parte izquierda del diagrama mostrado en la Ilustración 6, se distinguen los componentes *EntidadDGT*, *EntidadAC* y *EntidadRSU*. Estos componentes tienen como objetivo ejecutar las operaciones que necesitan cada una de las distintas entidades para desempeñar su labor en el sistema.

En la parte derecha del diagrama se ubica el componente *SoporteEntidades*, cuya misión es abstraer las operaciones comunes a los componentes comentados en el párrafo anterior. Este componente engloba una serie de subcomponentes, que se describen a continuación.

InformacionSimulador ofrece una serie de operaciones que permitan a las distintas entidades obtener información propia de las mismas necesarias para su funcionamiento, proveniente de la información asociada a la simulación en la cual se están ejecutando. Teniendo en cuenta el uso de NCTUns [Referencias-1] para la obtención de esta información, se muestra esta dependencia en el diagrama a través del uso de la interfaz *NCTUns* (de carácter lógico) por el componente *InformacionSimulador*.

Criptografia es el componente en el que se recoge la funcionalidad necesaria por las entidades para efectuar las distintas operaciones criptográficas del protocolo.

Log proporciona las operaciones necesarias para la gestión de logs. Como se puede observar en el diagrama, el componente *Criptografia* hace uso de este componente con el fin de almacenar en un log las distintas operaciones criptográficas efectuadas en el protocolo.

Comunicaciones es el componente que ofrece la funcionalidad que necesitan las entidades para comunicarse entre sí.

TomarMedidas ofrece operaciones a las entidades para que puedan obtener los indicadores de rendimiento del protocolo. Tal y como se puede comprobar, este componente hace uso del componente *Comunicaciones* para realizar el envío de los indicadores al sistema de representación de los mismos.

Como se puede comprobar, *EntidadRSU* hace uso del componente *ComunicacionInterfaz*, cuyo objetivo es ofrecer soporte a las RSU para que puedan transmitir información asociada al intercambio de los mensajes del protocolo con los vehículos, así como el contenido de sus *beacon*, al sistema de visualización de mensajes. Este componente, representado en color verde en el diagrama, no forma parte del ámbito de este Proyecto, sino que su desarrollo forma parte del Proyecto Fin de Carrera [Bibliografía-2] encargado de la simulación de la parte inalámbrica de EVIGEN [Bibliografía-1] (representada en el diagrama por medio del componente *ProtocoloEVIGEN_EntornoInalambrico*).

Dentro de *ProtocoloEVIGEN_EntornoInalambrico* se puede observar el componente denominado *ComponenteY*. Este componente presenta un carácter lógico, y su objetivo es mostrar en el diagrama los componentes de la simulación del entorno de infraestructura de EVIGEN [Bibliografía-1] que son utilizados por la parte inalámbrica desarrollada en el Proyecto paralelo [Bibliografía-2], los cuales se corresponden con los componentes *TomarMedidas* y *Criptografia*, para obtener los indicadores de rendimiento e incorporar mecanismos de seguridad, a la parte inalámbrica del protocolo.

3.3.1.2. Arquitectura preliminar del sistema de representación de indicadores.

Descrita la arquitectura preliminar de la parte del sistema encargada del desarrollo de la parte de infraestructura del protocolo, en la Ilustración 7 se presenta el diagrama de componentes correspondiente con el sistema de representación de indicadores.

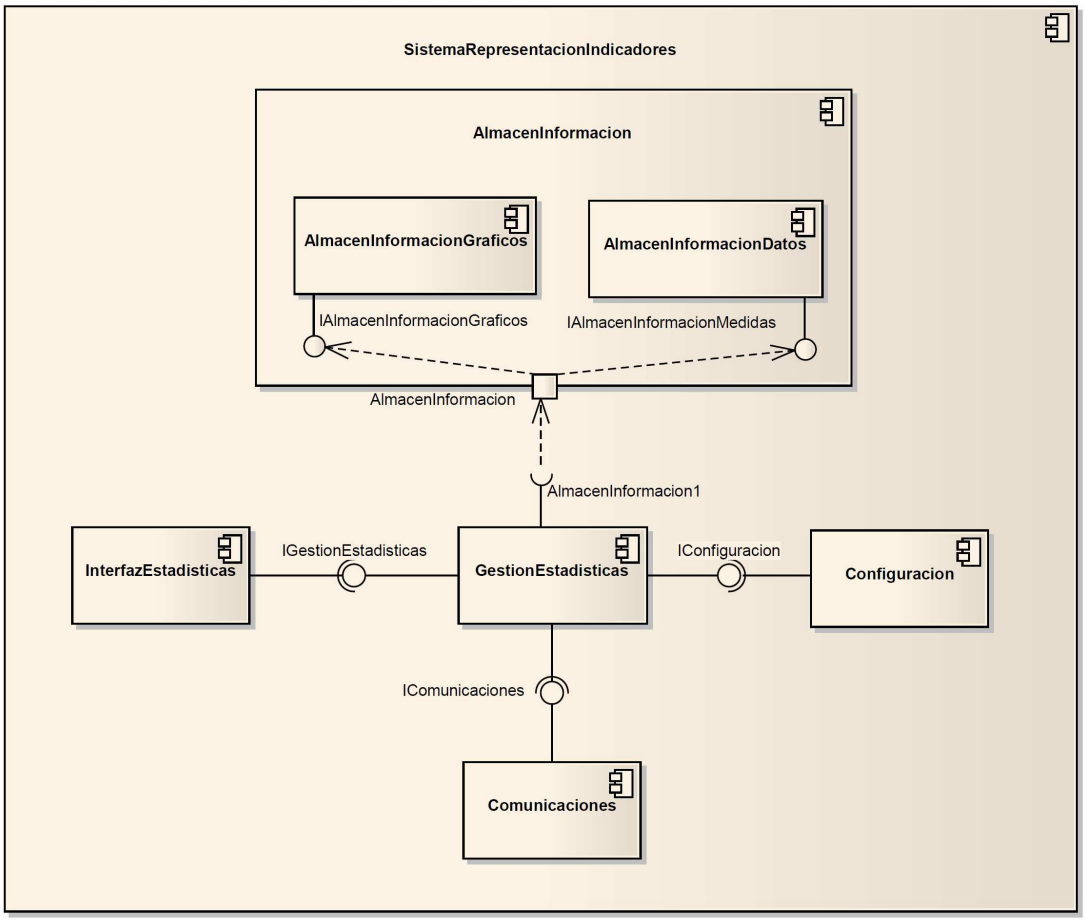

Ilustración 7: arquitectura preliminar del sistema de representación de indicadores.

El componente *InterfazEstadisticas* se corresponde con la interfaz gráfica en la cual se representan los distintos indicadores de rendimiento, y desde la cual el usuario puede gestionar el sistema.

GestionEstadisticas es el componente encargado de la recepción de los indicadores transmitidos desde las distintas entidades del protocolo, de realizar las operaciones de gestión del sistema, y de efectuar las operaciones de carga y almacenaje de la información en PDF o Excel 2003.

El componente *AlmacenInformacion* encapsula la funcionalidad relacionada con el almacenaje o carga de información en diferentes soportes de almacenamiento. Por este motivo, el componente *GestionEstadisticas* utiliza el mismo para realizar las operaciones de almacenamiento y carga en PDF o Excel 2003. Este componente está formado por los dos subcomponentes siguientes.

AlmacenInformacionGraficos proporciona funcionalidad para el almacenaje de estadísticas sobre unos datos determinados, en un dispositivo de almacenamiento concreto. En el sistema presente, la utilidad de este componente

es el de representar las estadísticas relacionadas con los indicadores de rendimiento en PDF.

AlmacenInformacionDatos ofrece operaciones para poder almacenar o cargar datos en distintos formatos de hojas de cálculo. Esto implica que, en el sistema que se desea crear, se haga uso de este componente para efectuar el almacenaje y carga de los datos de Excel 2003.

El componente *Comunicaciones* proporciona funcionalidad para la recepción de mensajes. Por ese motivo, el sistema hace uso de este componente para recibir los mensajes que contienen los diferentes indicadores de rendimiento de las entidades del protocolo.

Configuracion es un componente que ofrece operaciones para guardar o almacenar parámetros de configuración en diferentes dispositivos de almacenamiento. Por tanto, el sistema de representación de indicadores hace uso de este componente para guardar o cargar el valor del puerto por el cual se van a recibir los indicadores de rendimiento.

3.3.2. Análisis de las tecnologías impuestas.

En lo que respecta a las tecnologías que se imponen para el desarrollo del sistema, la principal se encuentra en la del uso del lenguaje C/C++ para Unix, para la implementación de las entidades de la infraestructura del protocolo EVIGEN [Bibliografía-1]. Esto se debe a que NCTUns [Referencias-1] está implementado en dicho lenguaje, y además, éste necesita ejecutarse sobre el sistema operativo Fedora 10 [Referencias-2], utilizando el kernel específico para el funcionamiento del simulador.

De esta restricción del lenguaje necesario para la creación del protocolo, surge otra imposición. Teniendo en cuenta que las distintas entidades del protocolo transmiten los indicadores que obtienen al sistema de representación de los mismos, la tecnología de comunicación elegida para la recepción de los indicadores debe ser compatible con la tecnología de comunicación destinada al envío de los mismos desde las entidades.

Otra imposición consiste en el uso del componente *ComunicacionInterfaz* para enviar la información relativa a los mensajes intercambiados entre las RSU y los vehículos, al sistema de visualización de mensajes. El motivo por el que se debe hacer uso de este componente, es debido a que su desarrollo forma parte del Proyecto Fin de Carrera destinado a la construcción de dicho sistema [Bibliografía-2], y en el cual se encapsula toda la funcionalidad necesaria para la realización de los envíos.

La última imposición se encuentra en el uso de las funciones proporcionadas por NCTUns [Referencias-1], para obtener la información propia de las entidades de los ficheros de configuración creados por el simulador en cada ejecución de una simulación, así como del kernel específico que utiliza el mismo. El motivo de esta imposición es que se debe hacer uso de las llamadas apropiadas al kernel de NCTUns [Referencias-1] para la obtención de la información, además que se evita tener que conocer el funcionamiento interno del simulador para la realización de dichas operaciones.

3.4. Análisis de seguridad.

En este apartado se realiza un análisis de seguridad con el fin de detectar las posibles amenazas y vulnerabilidades que se pueden encontrar en la ejecución de las distintas entidades que participan en el protocolo.

Es necesario indicar que dicho análisis no va a cubrir aspectos ya especificados en el artículo que describe el protocolo EVIGEN [Bibliografía-1], con el fin de centrarse únicamente en aquellos que aún no han sido analizados.

Como se puede comprobar, la vulnerabilidad principal reside en la utilización de las comunicaciones. Hay que tener en cuenta que el simulador NCTUns [Referencias-1] ejecuta todos los procesos en la máquina local, por lo que este problema no supondría un gran riesgo. Sin embargo, si se implantase este sistema en un entorno vehicular real, teniendo en cuenta el gran número de entidades que podrían participar, y por tanto, el gran número de comunicaciones que se producirían, el riesgo que podría suponer sería muy elevado. Por este motivo, teniendo en cuenta que el objetivo es su implantación algún día en entornos vehiculares reales, se va a realizar el análisis siguiendo este punto de vista.

En primer lugar, se estudia la posibilidad de la realización de ataques de denegación de servicio. Una de las suposiciones del protocolo es que las entidades DGT, AC y RSU tienen capacidad computacional infinita, por lo que el ataque a los recursos computacionales de las mismas no se va a considerar una amenaza. Sin embargo, se puede producir este ataque si los mensajes de entrada a las entidades son incorrectos, por lo que, con el fin de prevenir esta amenaza, las entidades deben comprobar que el formato y contenido de dichos mensajes son correctos antes de trabajar con su contenido.

En segundo lugar, resulta necesario indicar que ninguno de los mensajes transmitidos contiene ningún control de integridad, por lo que éstos podrían ser modificados sin tener conocimiento de ello. Esto supone un riesgo considerable, ya que si éstos son modificados durante su tránsito o por un atacante, pueden provocar graves problemas, como por ejemplo, no sancionar a vehículos infractores, sancionar a vehículos que no han cometido ninguna infracción, modificar el valor de los identificadores de los vehículos de la CRL de manera que se revoque el certificado de vehículos inocentes o no se revoque el de los vehículos infractores, etc.

Otro problema presente es la falta de autenticación y no repudio, lo cual implica un riesgo grave ya que cualquier atacante podría suplantar la identidad de otro, y además, se podría negar el envío de los correspondientes mensajes.

El problema de la integridad y no repudio se soluciona si las entidades firman los correspondientes mensajes antes de su transmisión. Por ello, se debe adjuntar la firma digital a todos los mensajes cuando éstos son creados (con excepción del *beacon* como se verá posteriormente). Además, con el objetivo de garantizar la autenticación, en el caso del mensaje con una notificación de infracción enviado desde una RSU, y el mensaje con el identificador de un vehículo infractor cuyo certificado se desea revocar enviado desde la DGT, se necesita adjuntar también el certificado de la entidad para que se pueda comprobar que el emisor es quien dice ser, y se pueda realizar la verificación de su firma. En cambio, en el mensaje con una notificación de sanción y en el mensaje con la CRL, no se adjunta el certificado

ya que las entidades a las que van dirigidos los mismos poseen los correspondientes certificados de la DGT y AC.

En lo que respecta a la confidencialidad de los mensajes, no se va a realizar el cifrado de ninguno de los mensajes del entorno de infraestructura del protocolo, por los motivos que se exponen a continuación:

- El mensaje con la notificación de sanción no puede ser cifrado, debido a que la DGT no dispone del certificado del vehículo infractor al que va dirigido.
- En lo que respecta al mensaje con la CRL, no debe ser cifrado ya que su contenido no va dirigido a un receptor concreto, sino que éste se propaga para que todos los vehículos se informen de los vehículos cuyo certificado ha sido revocado.
- El mensaje con la notificación de infracción de un vehículo no necesita ser cifrado, debido a que su contenido se transmite posteriormente en claro en la notificación de sanción correspondiente a dicha notificación de infracción.
- El mensaje que contiene el identificador del vehículo cuyo certificado se desea revocar no necesita ser cifrado, ya que dicho identificador se transmitirá contenido dentro de la CRL en claro.

Otra amenaza presente es la posibilidad de efectuar ataques de repetición, lo cual implica un riesgo grave en la seguridad de las comunicaciones. A continuación, se indica los mensajes en los que se puede explotar dicha vulnerabilidad:

- Mensaje con la notificación de infracción, ya que si un atacante se apodera de un mensaje de este tipo, puede provocar que se sancione de manera indefinida al vehículo infractor, enviando periódicamente el correspondiente mensaje a la DGT .
- Mensaje con la notificación de sanción, ya que si un atacante obtiene un mensaje de este tipo, puede estar continuamente sancionando al vehículo al que va dirigido la misma.

El mensaje con la notificación de sanción supone menos riesgo que el primero, ya que aunque se realizase el ataque, la DGT no tiene constancia del envío de dicha sanción, y por tanto, no puede aplicar ningún tipo de sanción al vehículo atacado, como por ejemplo revocar su certificado si no se recibe evidencia que la responde. El único inconveniente es que se ejecutaría el protocolo cada vez que se realizase el ataque, pudiendo, por ejemplo, provocar problemas de saturación en la red de comunicación así como en las propias entidades.

Con el fin de solucionar esta vulnerabilidad, se debe incluir para cada tipo de mensaje un identificador único, que no pueda volver a repetirse, de manera que si en una entidad cualquiera del protocolo se recibe uno de estos mensajes y se comprueba que previamente se ha recibido otro mensaje con el mismo identificador, este último es descartado. No obstante, con el segundo mensaje se tiene que tener en cuenta que, con el fin de intentar en la medida de lo posible conseguir el éxito del protocolo, las RSU deben propagar el mensaje con la notificación de sanción hasta que lo entreguen al vehículo infractor. Finalmente, resulta necesario destacar que para que esta solución pueda funcionar, la firma

también debe incluir al identificador, ya que de lo contrario se podría modificar el mismo, perdiéndose toda la efectividad de esta medida.

Además, en lo que respecta al mensaje con la notificación de sanción, si en la DGT se recibe un mensaje de evidencia de respuesta a una notificación de sanción, y esta entidad no ha enviado dicho mensaje, entonces el mensaje con la notificación de evidencia recibido debe ser descartado.

En lo que respecta al mensaje con la CRL, no se va a incluir ningún identificador ya que el carácter de este mensaje es el propagarse mediante inundación por toda la red, con el fin de informar a la gran mayoría posible de vehículos de su contenido. Tampoco se incluirá identificador al mensaje que contiene el identificador de un vehículo cuyo certificado se desea revocar, ya que en caso de producirse el ataque con este mensaje, la AC lo descartaría ya que el identificador se encontraría almacenado con anterioridad en la CRL.

Teniendo en cuenta lo comentado en el párrafo anterior, se puede presentar otro problema, aunque no es estrictamente de seguridad, y es el de la saturación de la red y de las entidades del protocolo, lo cual implica un gran riesgo, ya que de ser así no se podría realizar la ejecución del protocolo. Para ello, se debe incluir en el mensaje con la CRL un tiempo de vida, de manera que el mismo se vaya decrementando de manera unitaria en cada entidad por la que pase. Cuando este tiempo de vida expire (llegue a 0), debe ser descartado, consiguiendo así que dicho mensaje no esté propagándose de manera indefinida por la red.

Otro mensaje en el que sería útil incluir este tiempo de vida es en el mensaje de notificación de sanción, ya que como se comentó previamente, una RSU sólo puede descartar este mensaje si lo ha entregado previamente al vehículo infractor, por lo que en caso contrario debe propagarlo de manera indefinida por el resto de RSU vecinas hasta que pueda realizar la entrega. Esto implica un problema, ya que si una RSU nunca entrega este mensaje al vehículo infractor, se encontraría continuamente propagando dicho mensaje, lo cual puede suponer un fuerte impacto en los recursos de red. Por tanto, una solución a este problema es añadirle un tiempo de vida.

Finalmente, en lo que respecta a los *beacon* que las RSU envían a los vehículos, con el fin de que éstos puedan verificar su integridad y no repudio, las RSU deberían firmar los *beacon* antes de transmitirlos. Sin embargo, considerando la enorme frecuencia de transmisión de los *beacon*, y el gran número de vehículos que lo reciben, la realización de las operaciones criptográficas de firma y verificación puede suponer un fuerte impacto en el rendimiento de las entidades. Éste es un tema de discusión actual en congresos, y una de las soluciones propuestas es el envío de un *beacon* firmado cada cierto tiempo. No obstante, en el sistema a desarrollar no se va a implantar esta medida, pudiendo ser la misma implementada en ampliaciones futuras de este proyecto.

3.5. Estudio tecnológico.

En este apartado se hace un estudio de las diferentes tecnologías ya existentes, las cuales pueden servir como soporte para el desarrollo de los distintos componentes del sistema.

Al igual que se ha realizado en apartados anteriores, se va a dividir el estudio en dos partes correspondientes con las partes del sistema identificadas en el apartado 3.1, con el fin de no dificultar su explicación.

Antes de comenzar el estudio tecnológico, es necesario indicar que, debido a la gran cantidad de tecnología existente actualmente de distribución comercial, el estudio se va centrar principalmente en tecnología de distribución libre. El objetivo es evitar en la medida de lo posible un aumento del presupuesto en el aspecto tecnológico, teniendo en cuenta que siempre se obtenga menor coste en la utilización de tecnología de distribución libre y el posible mayor esfuerzo que requiera el desarrollo, que en la utilización de tecnología de distribución comercial que pueda facilitar el desarrollo.

3.5.1. Estudio tecnológico de la extensión de NCTUns para la simulación del entorno de infraestructura del protocolo EVIGEN.

Antes de describir en profundidad cada una de las tecnologías aplicables a cada uno de los componentes, se presenta en la Tabla 1, una tabla resumen del contenido de este apartado:

ESTUDIO TECNOLÓGICO INFRAESTRUCTURA EVIGEN		
Componente	**Necesidad tecnológica**	**Alternativas**
Criptografia	Soporte criptográfico para ofrecer servicios de seguridad al protocolo.	OpenSSL [Referencias-3].
		Crypto++ [Referencias-4].
		Botan [Referencias-5].
Comunicaciones	Mecanismos para la comunicación entre las entidades del protocolo.	Sockets TCP.
		Sockets UDP.
		CORBA [Referencias-6].
		ONC RPC.
		gSoap [Referencias-7].
		AXIS [Referencias-8].
Log	Soporte para almacenar entradas de log en un dispositivo de almacenamiento concreto.	Fichero.
		MySQL [Referencias-9].

Tabla 1: síntesis estudio tecnológico extensión NCTUns para simulación de entorno infraestructura de EVIGEN.

3.5.1.1. Tecnologías aplicables al componente Criptografia.

Teniendo en cuenta que este componente debe ser desarrollado en C/C++, se exponen aquellas tecnologías analizadas que mejor se pueden aplicar al mismo.

- OpenSSL [Referencias-3]: librería desarrollada en C que suministra una serie de funciones criptográficas de las que pueden hacer uso las aplicaciones. Ofrece soporte para gestión de certificados X509, y varios algoritmos de criptografía asimétrica y simétrica. Además, ofrece buena documentación y gran cantidad de ejemplos que facilitan el uso de esta librería.
- Crypto++ [Referencias-4]: librería escrita en C++ que proporciona un conjunto de algoritmos criptográficos que pueden ser utilizados por otros programas. Una de las ventajas de esta librería es que ofrece

multitud de algoritmos de criptografía asimétrica, criptografía simétrica y criptografía de curvas elípticas. No obstante, no ofrece soporte para certificados de clave pública, el API proporcionado es difícil de utilizar y la documentación que se encuentra de la misma es escasa.

- Botan [Referencias-5]: librería escrita en C++ que ofrece soporte para la utilización de diversos algoritmos de criptografía asimétrica y simétrica, así como funcionalidad para la gestión de certificados X509. Además, proporciona documentación y ejemplos de cómo utilizar esta librería.

3.5.1.2. Tecnologías aplicables al componente Comunicaciones.

Existen diversos mecanismos de comunicación que pueden ser implementados en C/C++. Entre estos mecanismos, los más apropiados para la implementación de la comunicación entre las entidades del protocolo, y la comunicación entre las entidades y el sistema de representación de indicadores, son los siguientes:

- Sockets: modo de comunicación independiente de la plataforma y del protocolo de comunicación establecido. Las ventajas de este mecanismo es que ofrece un buen rendimiento en la comunicación, y su diseño es bastante sencillo. Sin embargo, este mecanismo opera a nivel de transporte (TCP o UDP), siendo su programación a ese nivel, y por tanto, se tienen que realizar ciertas operaciones que otros mecanismos de comunicación abstraen, como por ejemplo, la codificación o decodificación (*marshalling* o *unmarshalling*) de la información que se envía y se recibe. Existen dos tipos de sockets:
  - TCP: es orientado a conexión, lo que proporciona medios que aseguran la recepción de los paquetes en el destino, la llegada y entrega ordenada de todos los paquetes, reenvíos de paquetes erróneos y paquetes perdidos, control de flujo y congestión, etc. Es más fiable que UDP, pero es más lento.
  - UDP: es no orientado a conexión, y no proporciona los medios descritos en TCP, teniendo que ser implementados a nivel de aplicación. Sin embargo, UDP es más rápido que TCP.
- CORBA [Referencias-6]: plataforma de desarrollo de sistemas distribuidos, basado en el paradigma de programación orientada a objetos, que permite la comunicación entre aplicaciones desarrolladas en diferentes lenguajes de programación y ejecutadas en máquinas con diferente arquitectura y sistema operativo. Además, esta plataforma abstrae la implementación de ciertas operaciones que se tienen que realizar en paradigmas de comunicación de más bajo nivel, como la codificación y decodificación de los mensajes, y la asignación de puertos, los cuales se establecen de forma transparente al programador a través del servicio de nombres. No obstante, la programación con CORBA [Referencias-6] resulta bastante laboriosa, ya que requiere del conocimiento de IDL, y aparte, se deben realizar numerosos pasos para la implementación de los clientes y servidores. Además, el rendimiento en la comunicación es menor que en otros modos de comunicación como sockets, debido a que los mensajes introducen mayor *overhead* y

se realizan mayor número de operaciones, con el fin de efectuar la comunicación entre las aplicaciones heterogéneas.

- ONC RPC: mecanismo de comunicación basado en RPC creado por Sun y con soporte para C/C++, Java y plataformas Windows, que permite, de manera transparente, la comunicación entre aplicaciones desarrolladas en estos distintos lenguajes. En cuanto a ventajas, al igual que CORBA, abstrae al programador de la realización de ciertas operaciones. Además, su programación no es tan costosa como en CORBA, aunque requiere del conocimiento del lenguaje XDR. Sin embargo, comparada con otras tecnologías como sockets, ofrece un menor rendimiento en la comunicación, debido al retardo producido en el empaquetamiento y desempaquetamiento de los datos en formato XDR. Otro inconveniente que presenta dicha tecnología, es que su desarrollo se centra principalmente en el paradigma cliente-servidor.

- gSoap [Referencias-7]: paquete de herramientas para el desarrollo de aplicaciones basadas en servicios web en C/C++. Ofrece mayor interoperabilidad que las otras tecnologías en la comunicación entre sistemas heterogéneos, debido a que ésta se basa en la aplicación de un conjunto de protocolos y estándares abiertos, permitiendo comunicación en redes tan heterogéneas como Internet. Sin embargo, su rendimiento en comparación con las otras tecnologías es muy bajo, teniendo en cuenta que la información a enviar se debe describir en XML, se debe empaquetar en SOAP y se debe enviar sobre HTTP. Aparte, el objetivo de servicios web se centra principalmente en ofrecer servicios a través de Internet, por lo que se suelen ejecutar en servidores o contenedores web (aunque se pueden ejecutar sobre la propia máquina), y su diseño está orientado hacia el paradigma cliente-servidor.

- AXIS [Referencias-8]: implementación de SOAP, con soporte para Java y C++, que proporciona un entorno de desarrollo y ejecución de servicios web. Las ventajas e inconvenientes de esta tecnología son similares a los expuestos en gSoap [Referencias-7].

3.5.1.3. Tecnologías aplicables al componente Log.

A continuación, se exponen una serie de tecnologías que se pueden aplicar para la creación del log:

- Fichero: fichero de texto normal en el que se van almacenando las entradas del log. Esta tecnología presenta la ventaja de que, con las interfaces proporcionadas por los sistemas operativos Unix para la inserción de elementos en ficheros, su desarrollo no es complicado. Además, con el uso de un fichero se permite, en sistemas operativos Unix, utilizar el comando *tail -f* para ir observando las entradas del log conforme se van insertando en el mismo. Sin embargo, presenta el inconveniente de que varios procesos accediendo concurrentemente al fichero podrían provocar resultados inesperados, por lo que las medidas para evitarlo se tendrían que programar por separado.

- MySQL [Referencias-9]: sistema de gestión de base de datos relacional, que utiliza el lenguaje SQL, y que ofrece soporte para el acceso a las

bases de datos desde diferentes lenguajes de programación como Java, C#, C, C++, etc. La ventaja que presenta esta tecnología es que ofrece soporte multiusuario, por lo que el acceso concurrente al log sería controlado de manera transparente al programador. No obstante, teniendo en cuenta que en el log únicamente se van a realizar inserciones al final del mismo, el uso de una interfaz de base de datos para programarlo puede resultar una tarea laboriosa. Además, de esta manera, ya no se puede hacer uso del comando *tail -f*, lo cual, teniendo en cuenta el carácter del log, puede suponer un inconveniente.

Aunque existen muchas bases de datos, en este estudio se especifica la base de datos MySQL [Referencias-9], debido a que es de tipo relacional, de distribución libre, porque utiliza el lenguaje SQL y ofrece soporte para distintos lenguajes de programación.

3.5.2. Estudio tecnológico del sistema de representación de indicadores.

Al igual que se ha hecho en el subapartado anterior, previamente a la exposición de las distintas tecnologías aplicables a cada uno de los componentes de esta parte del sistema, en la Tabla 2 se presenta una síntesis de las mismas:

ESTUDIO TECNOLÓGICO SISTEMA REPRESENTACIÓN INDICADORES		
Componente	**Necesidad tecnológica**	**Alternativas**
InterfazEstadisticas	Herramientas para el diseño de interfaces gráficas.	Windows Forms.
		Java Swing.
		GTK+ [Referencias-10].
	Soporte para la creación de estadísticas.	Java2D [Referencias-11].
		JFreeChart [Referencias-12].
		Graphics.
AlmacenInformacionGraficos	Soporte para el almacenaje de información en formato PDF.	iText [Referencias-13].
		iTextSharp [Referencias-13].
AlmacenInformacionDatos	Soporte para el almacenaje y carga de datos en formato Excel 2003.	Microsoft Excel Object Library.
		JExcelApi [Referencias-14].
Configuracion	Soporte para la obtención de parámetros de configuración de un dispositivo de almacenamiento determinado.	Fichero.
		MySQL [Referencias-9].
Comunicaciones	Mecanismo de comunicación entre las entidades del protocolo y el sistema de representación de indicadores.	Sockets TCP.
		Sockets UDP.
		CORBA [Referencias-6].
		ONC RPC.
		AXIS [Referencias-8].

Tabla 2: síntesis estudio tecnológico sistemas representación indicadores.

3.5.2.1. Tecnologías aplicables al componente InterfazEstadisticas.

En primer lugar, se comentan las tecnologías que ofrecen mejores medios para facilitar el desarrollo de interfaces gráficas:

- Windows Forms: librería para el desarrollo de interfaces gráficas incluida dentro del *framework* de Microsoft .NET. Además, utilizando entornos de desarrollo como Microsoft Visual Studio [Referencias-15], la creación de las interfaces no presenta gran dificultad. No obstante, este entorno es de distribución comercial y la interfaz gráfica creada sólo funcionaría en sistemas operativos Windows. Por ello, otra opción es la utilización del Proyecto Mono [Referencias-16] para el desarrollo de la interfaz, ya que permite la creación de aplicaciones compatibles con distintos sistemas operativos y arquitecturas, y además, es de distribución libre.
- Java Swing: biblioteca para el desarrollo de interfaces gráficas en Java, independientes de la plataforma. Además, utilizando entornos de desarrollo como el proporcionado por NetBeans [Referencias-17], la creación de interfaces gráficas resulta bastante sencilla.

- GTK+ [Referencias-10]: conjunto de librerías multiplataforma con soporte para Linux, Windows y Mac, que permite la creación de interfaces gráficas en distintos lenguajes de programación como C, C++, C#, PHP, Ruby, etc.

No se van a incluir en el estudio tecnologías destinadas al desarrollo de interfaces web, ya que su diseño está más bien orientado hacia el paradigma petición-respuesta, mostrando la información correspondiente a la petición realizada por un determinado cliente. Teniendo en cuenta el carácter de esta interfaz, que estará siendo constantemente actualizada debido a los envíos de los indicadores, se considera mejor opción el uso de las tecnologías que han sido expuestas.

Teniendo en cuenta que en la interfaz se debe mostrar una serie de gráficas con el valor de los indicadores de rendimiento, es conveniente estudiar la distinta tecnología existente para la creación de las mismas:

- Java2D [Referencias-11]: biblioteca que proporciona soporte para la creación de gráficos 2D en Java. Como ventajas presenta que los gráficos que se generan tienen buena presencia y se dispone de bastante documentación sobre su uso. Además, Java Swing ofrece soporte para dicha tecnología.
- JFreeChart [Referencias-12]: librería, desarrollada en Java, utilizada para la creación de múltiples tipos de gráficos. Esta librería presenta como ventajas que los gráficos generados tienen una buena presentación, que dispone de bastante documentación que facilita su uso, así como gran variedad de ejemplos, y que puede ser combinada fácilmente con librerías para la creación de PDF (como iText [Referencias-13]). Además, esta tecnología presenta soporte para su integración con Java Swing.
- Graphics: biblioteca incluida dentro del *framework* de .NET, la cual ofrece soporte para la creación de gráficos en los distintos lenguajes de esta plataforma. Dispone de buena documentación sobre su utilización, y los gráficos creados presentan buena calidad.

3.5.2.2. Tecnologías aplicables al componente AlmacenInformacionGraficos.

En este punto, se exponen las tecnologías estudiadas que mejor soporte proporcionan para el almacenaje de la información en PDF:

- iText [Referencias-13]: librería que proporciona funcionalidad para la manipulación de archivos PDF en Java. Esta biblioteca ofrece una interfaz sencilla para la gestión de archivos PDF, además de disponer de documentación y ejemplos de uso que facilitan en gran medida su utilización. También proporciona soporte para el almacenaje de gráficos procedentes de la librería JFreeChart [Referencias-12], y se tienen bastantes ejemplos y documentación sobre cómo hacerlo.
- iTextSharp [Referencias-13]: biblioteca que ofrece una interfaz sencilla para la gestión de archivos PDF en C#. Teniendo en cuenta que dicha librería es la versión en C# de iText [Referencias-13], ofrece sus mismas ventajas. No obstante, no se tiene demasiada información de cómo almacenar gráficos en PDF utilizando esta librería.

3.5.2.3. Tecnologías aplicables al componente AlmacenInformacionDatos.

En lo que respecta a las tecnologías que ofrecen mejor soporte para el almacenaje y carga de los datos en formato Excel 2003, se han optado por las siguientes:

- Microsoft Excel Object Library: conjunto de librerías para .NET, que proporcionan varias operaciones para la manipulación de archivos en formato Excel. Su uso no presenta gran dificultad, y se dispone de cierta documentación y ejemplos sobre cómo utilizarla, especialmente para C# y Visual Basic.
- JExcelApi [Referencias-14]: librería desarrollada en Java, que ofrece soporte para la gestión de ficheros en formato Excel en dicho lenguaje. Esta biblioteca proporciona una interfaz no muy complicada para la manipulación de los archivos, y ofrece una documentación muy completa, así como varios ejemplos, que facilitan mucho su utilización.

3.5.2.4. Tecnologías aplicables al componente Configuracion.

En este punto, se analizan las mismas tecnologías que en el subapartado 3.5.1.3, pero estudiando las ventajas e inconvenientes que tienen para la obtención y establecimiento de los parámetros de configuración:

- Fichero: una de las ventajas de esta tecnología es que la gran mayoría de los lenguajes de programación proporcionan interfaces sencillas para la manipulación de ficheros. Sin embargo, se deben programar todas las operaciones de actualización y obtención de los parámetros de configuración.
- MySQL: con esta tecnología ya no se necesita actualizar u obtener directamente del soporte de almacenamiento la información, limitándose únicamente el programador a especificar las operaciones de actualización y consulta a la base de datos . No obstante, se necesita conocer el lenguaje utilizado por el gestor de la base de datos, el rendimiento de utilizar una base de datos es menor, y además, la instalación de una base de datos consume mayor número de recursos del computador.

3.5.2.5. Tecnologías aplicables al componente Comunicaciones.

Tal y como se comentó en el apartado 3.3.2, teniendo en cuenta que el sistema de representación de los indicadores de rendimiento recibe los indicadores de las entidades del protocolo desarrolladas en C/C++, la tecnología elegida en este componente debe ser compatible con la tecnología del componente *Comunicaciones* de la parte del sistema encargada de la implementación de las entidades del entorno de infraestructura de EVIGEN [Bibliografía-1].

3.6. Arquitectura definitiva de alto nivel y selección de tecnologías.

En este apartado se presentan los diagramas de componentes que muestran la arquitectura definitiva de alto nivel del sistema, y posteriormente, se indican las tecnologías que se han seleccionado del estudio tecnológico efectuado en el apartado 3.5.

3.6.1. Arquitectura de alto nivel definitiva.

En este punto, se muestra la arquitectura definitiva de alto nivel del sistema. Teniendo en cuenta las dos partes en las que se divide el sistema descritas en el apartado 3.2, en los subapartados siguientes se presentan dos diagramas de componentes correspondientes con cada una de ellas.

3.6.1.1. Arquitectura de alto nivel definitiva de la extensión de NCTUns para la simulación del entorno de infraestructura del protocolo EVIGEN.

En la Ilustración 8, se presenta el diagrama de componentes que muestra la arquitectura utilizada para la extensión de NCTUns [Referencias-1] para la simulación de las entidades de la infraestructura de EVIGEN [Bibliografía-1].

Como se puede observar, en la parte izquierda del diagrama se encuentran los componentes *EntidadDGT*, *EntidadAC* y *EntidadRSU*. Cada uno de estos componentes se encarga de la realización de las operaciones que necesita cada entidad para especificar su comportamiento dentro del sistema.

En la parte derecha del diagrama se encuentra el componente *SoporteEntidades*, cuyo objetivo es el de proporcionar funcionalidad a las diferentes entidades para facilitar la realización de las operaciones que definen su comportamiento Como se puede apreciar en el diagrama, los componentes correspondientes a las entidades hacen uso del componente *SoporteEntidades* a través del puerto del mismo nombre.

Tal y como se observa, el componente *SoporteEntidades* contiene una serie de subcomponentes, los cuales tienen como fin el desarrollo de una parte de la funcionalidad de soporte. En los párrafos expuestos posteriormente, se describe el objetivo de cada uno de estos subcomponentes.

InformacionSimulador es un componente encargado de obtener información necesaria por las entidades para su correcta ejecución, la cual es generada por NCTUns [Referencias-1] en cada simulación. Esta información puede ser, por ejemplo, el identificador de las RSU, la posición de las entidades, etc. Como se puede observar, este componente hace uso de la interfaz lógica *NCTUns*, con el objetivo de ilustrar que el componente *InformacionSimulador* interactúa con el simulador para obtener la información correspondiente.

GestionProtocolo es un componente que abstrae a las entidades de la realización de ciertas operaciones, como pueden ser, la creación de los distintos mensajes, la decodificación de los mensajes recibidos, la realización de las operaciones criptográficas del protocolo, etc.

Criptografia es el componente en el que se implementan las distintas operaciones criptográficas de las que van a hacer uso las distintas entidades del protocolo para su desarrollo. Como se aprecia en el diagrama, el componente

GestionProtocolo hace uso de este componente para la realización de las operaciones criptográficas específicas del protocolo.

Teniendo en cuenta que uno de los objetivos de este Proyecto es la creación de un módulo encargado de ofrecer distintas operaciones criptográficas, que puedan ser utilizadas para ofrecer determinados servicios de seguridad sobre el protocolo, el componente *Criptografia* debe ser diseñado e implementado de manera que cumpla con dicho objetivo.

Log es un componente que proporciona funcionalidad para la creación de logs. Como se observa, el componente *Criptografia* utiliza este componente ya que debe registrar todas las operaciones criptográficas realizadas por medio del mismo.

TomarMedidas es el componente utilizado por las entidades para obtener los indicadores de rendimiento del protocolo.

ObtencionConfiguracion tiene como objetivo proporcionar funcionalidad a las entidades para que puedan obtener las distintas variables de configuración que necesitan para funcionar, de un dispositivo de almacenamiento concreto. Además, el componente *TomarMedidas* hace uso de éste para leer los parámetros que necesita para comunicarse con el sistema de representación de indicadores.

Comunicaciones es el componente que abstrae todos los mecanismos de comunicación que pueden utilizar las entidades para comunicarse, como por ejemplo sockets UDP. Como se refleja en el diagrama de componentes, este componente ofrece dos interfaces: *IRealizacionComunicaciones* e *IGestionComunicacion*. La primera de ellas proporciona las operaciones que necesitan las entidades para establecer comunicación con otras o para escuchar en determinados puertos, mientras que *IGestionComunicacion* se utiliza una vez establecida la comunicación para realizar los envíos y recepciones de los mensajes. Resulta necesario indicar que el componente *TomarMedidas* también hace uso de este componente para realizar el envío de los indicadores al sistema de representación de los mismos.

Comparando este diagrama con el diagrama de componentes mostrado en la Ilustración 6 del apartado 3.3.1.1, se puede observar que el puerto *SoporteEntidades* no hace uso de las interfaces *IComunicaciones* e *IComunicacion*. Esto se debe a que el comportamiento de este puerto no es compatible con estas interfaces, debido a que una entidad puede tener una única instancia del puerto, común para todos los hilos creados en ella. Sin embargo, el comportamiento del componente *Comunicaciones* implica la necesidad de tener varias instancias de sus interfaces para cada una de las entidades, ya que se pueden mantener a la vez varias comunicaciones, y por tanto, no se puede utilizar una interfaz común para todas.

Como ya se indicó en el apartado 3.3.1.1, el componente *ComunicacionInterfaz*, el cual está representado en el diagrama en color verde, no entra dentro del ámbito de este Proyecto, y su objetivo es servir de soporte a las RSU para enviar al sistema de visualización de mensajes, la información asociada al intercambio de los mensajes con los vehículos, así como el contenido de sus *beacon*. Al igual que se especificó en la arquitectura preliminar del sistema, este componente es desarrollado por el otro Proyecto

Fin de Carrera [Bibliografía-2], cuyo uno de sus objetivos es la extensión de NCTUns [Referencias-1] para la simulación del entorno inalámbrico del protocolo EVIGEN [Bibliografía-1] (representado en el diagrama por medio del componente *ProtocoloEVIGEN_EntornoInalambrico*).

Finalmente, en el interior del componente *ProtocoloEVIGEN_EntornoInalambrico* se puede distinguir el componente lógico *ComponenteY*, el cual, tal y como se indicó en el apartado 3.3.1.1, representa la parte de la simulación del entorno inalámbrico de EVIGEN [Bibliografía-1] que utiliza componentes del entorno de infraestructura desarrollado en el presente Proyecto.

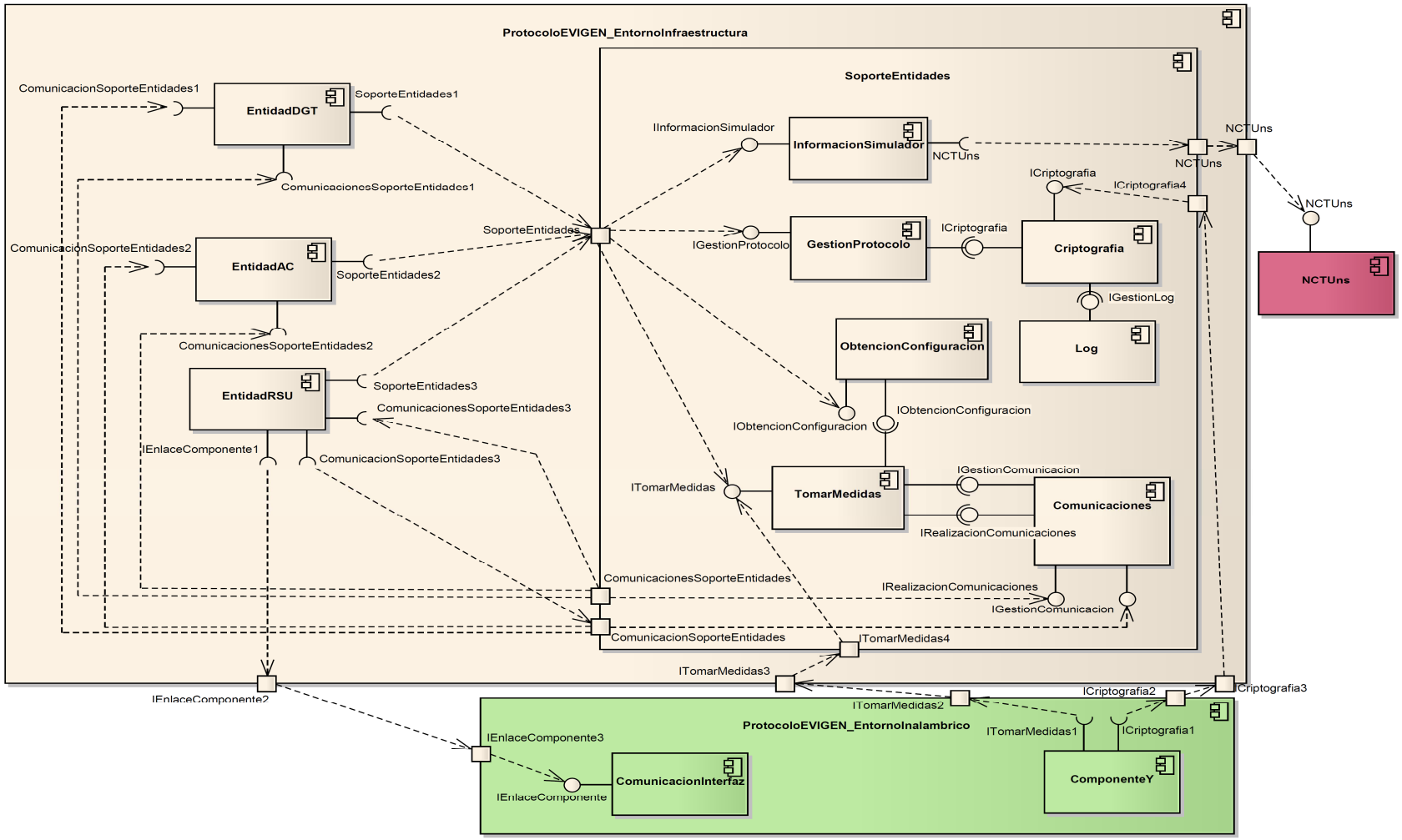

Ilustración 8: arquitectura definitiva de extensión de NCTUns para simulación de entidades de infraestructura de EVIGEN.

3.6.1.2. Arquitectura de alto nivel definitiva del sistema de representación de indicadores.

Una vez descrito el diagrama de componentes correspondiente con la extensión del simulador para la simulación del entorno de infraestructura de EVIGEN [Bibliografía-1], en la Ilustración 9 se presenta el diagrama de componentes del sistema encargado de la representación de los indicadores de rendimiento.

Previamente a la descripción del diagrama, es necesario indicar que para realizar la arquitectura de la parte del sistema encargado de la representación de los indicadores de rendimiento, se ha hecho uso del estilo arquitectónico Modelo-Vista-Controlador (MVC) [Bibliografía-3]. El motivo por el que se ha efectuado esta elección se debe a que la funcionalidad de este sistema puede ser separada en dos partes con objetivos distintos. Por un lado, la interfaz gráfica (Vista) en la que se va a reflejar la información de rendimiento y la utilizada por los usuarios para la interacción con el sistema, y por otro lado, la parte del sistema en la que se ubica toda la lógica de negocio del mismo (Modelo). También es conveniente el uso de un intermediario (Controlador) encargado de ejecutar las operaciones correspondientes del Modelo, en función de las acciones realizadas por los usuarios en la interfaz (Vista). Además, también se permite interacción entre Modelo y Vista, con el objetivo de mostrar en la Vista cualquier cambio producido en el Modelo, tal y como ocurre en el sistema para mostrar constantemente el rendimiento actualizado obtenido a partir de los indicadores. Por tanto, este estilo define una buena solución para la realización del diseño arquitectónico de esta parte del sistema.

Una vez explicado el estilo arquitectónico empleado, se procede a la descripción de esta arquitectura.

El componente *VistaEstadisticas* se corresponde con la Vista del estilo MVC [Bibliografía-3], y se encarga de la creación de la interfaz gráfica del sistema desde la que el usuario puede gestionar el mismo, y en la que se representan los distintos indicadores de rendimiento. Este componente ofrece la interfaz *IPintarEstadisticas*, utilizada para la representación de los indicadores.

El componente *ControladorEstadisticas* se corresponde con el Controlador del estilo MVC [Bibliografía-3], y se encarga de recibir los eventos de la interfaz para hacer las llamadas correspondientes al Modelo, y de comprobar que los parámetros de entrada a la misma son correctos, antes de dirigirlos hacia el Modelo.

El componente *ModeloEstadisticas* se corresponde con el Modelo de MVC [Bibliografía-3], ya que éste contiene toda la lógica de negocio del sistema de representación de indicadores. Tal y como se contempla en el diagrama, este componente se encuentra compuesto por un conjunto de subcomponentes, los cuales se describen en las líneas posteriores.

RealizacionEstadisticas tiene como objetivo la recepción de los indicadores procedentes de las diferentes entidades del protocolo y de representarlos en la interfaz gráfica haciendo uso de la interfaz proporcionada por el componente *VistaEstadisticas*, *IPintarEstadisticas*, de acuerdo a la idea proporcionada por el patrón *Observer* [Bibliografía-3]. Además, este componente se encarga de

las operaciones de gestión del sistema, y de almacenar y cargar la información correspondiente en PDF o Excel 2003.

AlmacenInformacion es un componente que engloba a todos aquellos componentes destinados al almacenaje de estadísticas en distintos soportes electrónicos, y al almacenaje y carga de datos en formatos diferentes de hojas de cálculo. Como se puede comprobar, el componente *RealizacionEstadisticas* hace uso de éste como soporte para la implementación de las operaciones de almacenamiento y carga en PDF o Excel 2003. Está formado por los tres subcomponentes especificados en los párrafos siguientes.

CargarInformacionDatos ofrece mecanismos utilizados para obtener datos almacenados en formatos distintos de hojas de cálculo. En el sistema que se desea desarrollar, se utiliza este componente para la carga de los datos de los indicadores de rendimiento de Excel 2003.

GuardarInformacionDatos proporciona funcionalidad suficiente para poder guardar los datos en formatos distintos de hojas de cálculo. En el sistema de representación de indicadores, se hace uso de este componente para el almacenaje de los indicadores de rendimiento en formato Excel 2003.

GuardarInformacionGraficos ofrece la funcionalidad que se necesita para poder representar estadísticas asociadas a unos datos concretos, en un dispositivo de almacenamiento determinado. Este componente es utilizado por el sistema para reflejar las estadísticas creadas a partir de los indicadores de rendimiento en formato PDF.

El componente *Comunicaciones* se encarga de abstraer la implementación de ciertos mecanismos de comunicación, utilizados para la recepción de mensajes. Como se puede verificar, el componente *RealizacionEstadisticas* utiliza este componente para recibir el valor de los indicadores de rendimiento de las diferentes entidades del protocolo.

Configuracion es un componente que ofrece operaciones para guardar o cargar parámetros de configuración en distintos soportes de almacenamiento. En el sistema a crear, se hace uso de este componente para almacenar o cargar el puerto por el que se van a recibir los mensajes con los indicadores. Dicho componente está compuesto por dos subcomponentes distintos.

LecturaConfiguracion ofrece las operaciones para cargar los parámetros de configuración. Por tanto, se hace uso de este componente para obtener el puerto del dispositivo de almacenamiento determinado.

EscrituraConfiguracion proporciona la funcionalidad para el almacenaje de los valores de configuración. Por este motivo, en el sistema a desarrollar se utiliza este componente para guardar el valor del puerto en el sistema de almacenamiento deseado.

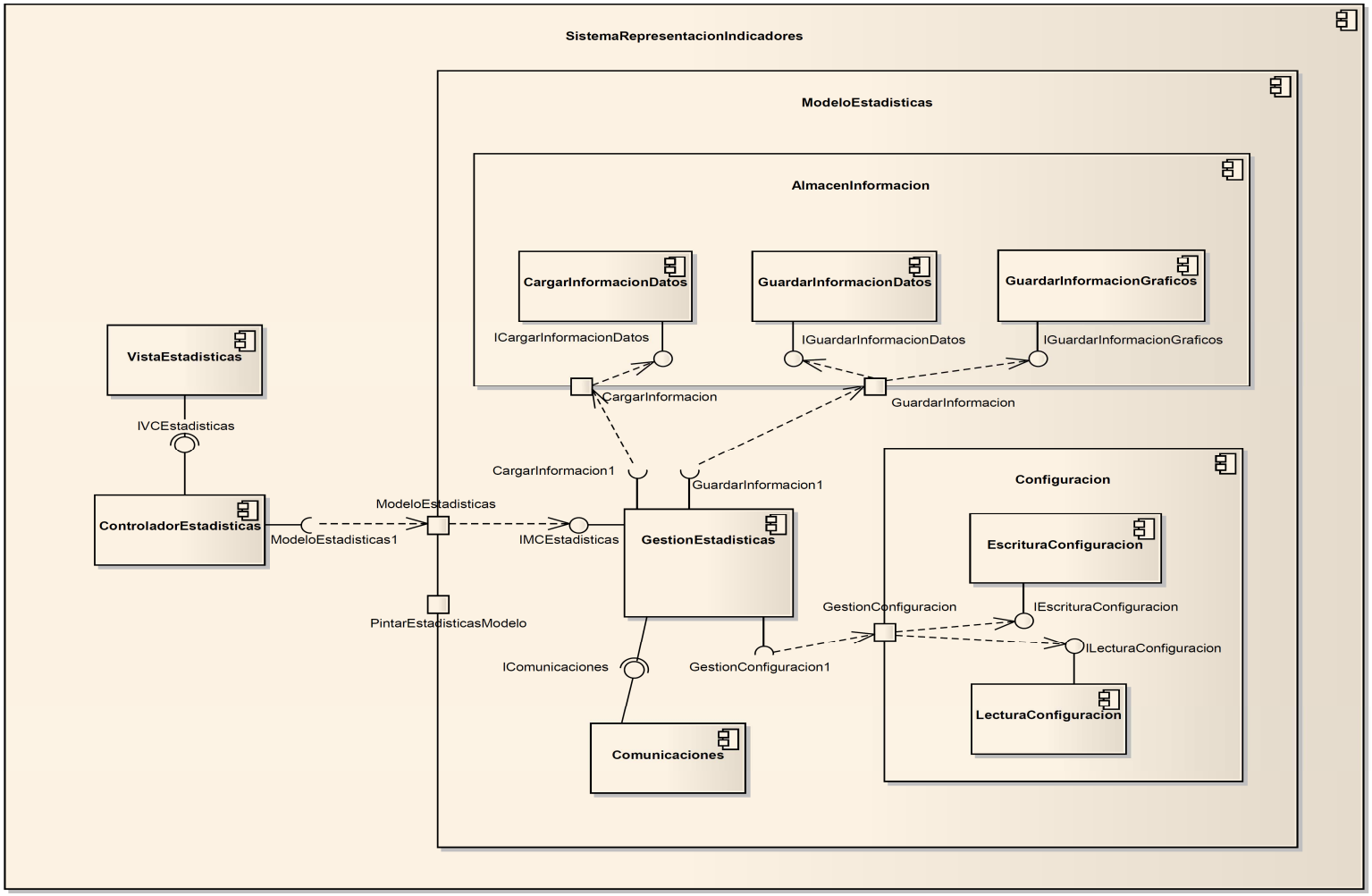

Ilustración 9: arquitectura definitiva del sistema de representación de indicadores.

3.6.2. Selección de las tecnologías no impuestas.

Teniendo en cuenta el estudio tecnológico realizado en el apartado 3.5, en los subapartados siguientes se especifica la tecnología que ha sido finalmente seleccionada.

3.6.2.1. Selección de las tecnologías no impuestas para la extensión de NCTUns para la simulación del entorno de infraestructura del protocolo EVIGEN.

En la Tabla 3 se presenta la tecnología seleccionada de cada uno de los componentes analizados en el estudio tecnológico, de la extensión de NCTUns [Referencias-1] para la simulación de las entidades de la infraestructura del protocolo.

SELECCIÓN TECNOLOGÍA INFRAESTRUCTURA EVIGEN		
Componente	**Necesidad tecnológica**	**Alternativas**
Criptografia	Soporte criptográfico para ofrecer servicios de seguridad al protocolo.	OpenSSL [Referencias-3].
Comunicaciones	Mecanismos para la comunicación entre las entidades del protocolo.	Sockets UDP.
Log	Soporte para almacenar entradas de log en un dispositivo de almacenamiento concreto.	Fichero.

Tabla 3: tecnologías seleccionadas para la extensión de NCTUns para simulación de entidades de infraestructura de EVIGEN.

En los párrafos siguientes se describen los motivos por los que ha sido seleccionada cada tecnología.

Para la implementación de la criptografía se va a hacer uso de OpenSSL [Referencias-3] ya que proporciona soporte para diversos algoritmos de criptografía simétrica y asimétrica, y mejor documentación y mayor número de ejemplos que otras bibliotecas como Botan [Referencias-5].

En lo que respecta a las comunicaciones entre las entidades, se va a hacer uso de sockets, con el fin de dar mayor importancia al rendimiento en las comunicaciones frente a una mayor abstracción en la programación. El motivo de esta elección se debe principalmente al coste de rendimiento ya presente en la ejecución del simulador NCTUns [Referencias-1], debido a que ejecuta todos los procesos correspondientes con cada una de las entidades que participan en el protocolo en la máquina local. Esto puede provocar que los indicadores de rendimiento que se obtengan del protocolo difieran en gran medida de los que se obtendrían de realizar las pruebas en un entorno vehicular real. Por tanto, con el fin de compensar la falta de rendimiento, se propone esta tecnología.

Otro motivo de esta elección es que los sockets se pueden utilizar en multitud de paradigmas de comunicación, a diferencia de otras tecnologías como RPC o servicios web cuyo diseño está más pensado para el paradigma cliente-servidor. Además, la necesidad de aplicar criptografía a los distintos mensajes del protocolo implica su previa codificación, por lo que otro de los motivos de utilizar sockets es que no se obtiene ninguna ventaja en la abstracción del empaquetamiento de los datos que ofrecen otras tecnologías de más alto nivel.

Teniendo en cuenta los dos tipos de sockets existentes, lo apropiado sería utilizar TCP para asegurar la fiabilidad en las comunicaciones. Sin embargo, esta fiabilidad supone un decremento del rendimiento (control de errores, control de retransmisiones, mayor número de mensajes, control de flujo, etc.), y además, al ser un protocolo orientado a conexión y teniendo en cuenta que todos los procesos ejecutan en la misma máquina, el número de conexiones existentes que se deben mantener en la misma puede suponer un impacto grave en los recursos y rendimiento de la máquina. Por esa razón, y considerando que el objetivo primordial es evaluar el rendimiento del protocolo, se va a optar por el uso de UDP.

En lo que respecta a la creación del log, la mejor opción es utilizar un fichero, ya que la inserción en el mismo no va a suponer un gran problema, y sobre todo, permite el uso del comando *tail -f* para ir mostrando las entradas del mismo conforme se van añadiendo, lo cual puede servir de utilidad teniendo en cuenta el carácter de los logs. Sin embargo, es necesario recordar que esta tecnología implica la programación de las medidas que eviten el acceso concurrente al mismo por parte de varios procesos.

3.6.2.2. Selección de las tecnologías no impuestas para el desarrollo del sistema de representación de indicadores.

Al igual que en el subapartado anterior, en la Tabla 4 se muestran las tecnologías seleccionadas para la implementación del sistema de representación de indicadores.

ESTUDIO TECNOLÓGICO SISTEMA REPRESENTACIÓN INDICADORES		
Componente	**Necesidad tecnológica**	**Alternativas**
InterfazEstadisticas	Herramientas para el diseño de interfaces gráficas.	Java Swing.
	Soporte para la creación de estadísticas.	JFreeChart [Referencias-12].
AlmacenInformacionGraficos	Soporte para el almacenaje de información en formato PDF.	iText [Referencias-13].
AlmacenInformacionDatos	Soporte para el almacenaje y carga de datos en formato Excel 2003.	JExcelApi [Referencias-14].
Configuracion	Soporte para la obtención de parámetros de configuración de un dispositivo de almacenamiento determinado.	Fichero.
Comunicaciones	Mecanismo de comunicación entre las entidades del protocolo y el sistema de representación de indicadores.	Sockets UDP.

Tabla 4: tecnologías seleccionadas para el desarrollo del sistema de representación de indicadores.

Una vez expuesta la tabla, en los siguientes párrafos se justifica el uso de dicha tecnología.

En cuanto a la tecnología utilizada para el almacenaje de las estadísticas de rendimiento del protocolo en PDF, se va a optar por el uso de iText [Referencias-

13] debido a que ofrece una interfaz sencilla para la manipulación de ficheros PDF, a la gran cantidad de documentación y ejemplos de uso, y a la buena documentación y ejemplos existentes de cómo almacenar los gráficos creados por medio de la biblioteca JFreeChart [Referencias-12] en PDF.

De la selección anterior se asume la utilización de JFreeChart [Referencias-12] para la representación de las gráficas de los indicadores de rendimiento, y del uso de Java Swing para la interfaz gráfica, debido a que JFreeChart [Referencias-12] ofrece soporte para esta tecnología.

Además, en lo que respecta a la tecnología destinada al almacenaje y carga de los indicadores de Excel 2003, teniendo en cuenta las mejores opciones incluidas tras el estudio realizado, y que el lenguaje de programación en el que se va a desarrollar este sistema es Java, la opción elegida es JExcelApi [Referencias-14].

En cuanto a la comunicación entre las entidades del protocolo implementadas en C/C++ y el sistema de representación de indicadore desarrollado en Java, considerando que Java ofrece un API para la creación de sockets, y que esta comunicación debe ser lo más rápida posible para que los indicadores se presenten cuanto antes en el sistema de representación, afectando en lo más mínimo al rendimiento del protocolo, se va a hacer uso de sockets UDP para dicha comunicación.

Finalmente, en lo que respecta a las tecnologías relacionadas con la obtención y establecimiento de la configuración del sistema de representación de indicadores, teniendo en cuenta que la única variable de configuración que se gestiona es el valor del puerto de recepción de los indicadores, se cree innecesario la instalación de una base de datos, siendo la gestión por medio de una interfaz de manipulación de ficheros sencilla de realizar.

3.7. Diagrama de casos de uso.

En esta sección se exponen los casos de uso del sistema, con el propósito de facilitar la obtención de los requisitos de este Proyecto.

3.7.1. Actores del sistema.

- *Usuario NCTUns*: actor encargado de interactuar directamente con NCTUns [Referencias-1], para efectuar las simulaciones del funcionamiento de las distintas entidades de EVIGEN [Bibliografía-1], en distintos escenarios de simulación.
- *Usuario sistema representacion indicadores*: actor encargado de reflejar los indicadores de rendimiento en el sistema de representación de los mismos. También se encarga de almacenar los indicadores en formato PDF, y de almacenar y cargar los mismos en formato Excel 2003.

3.7.2. Formato de plantilla de caso de uso.

El formato para la especificación de los casos de uso se presenta en la Tabla 5:

CASO DE USO	
Título:	
Identificador:	
Versión:	**Fuente:**
Actores:	
Objetivo:	
Precondiciones:	
Postcondiciones:	
ESCENARIO BÁSICO	
ESCENARIO ALTERNATIVOS	
Escenario 1	
Escenario 2	
Escenario N	

Tabla 5: plantilla de Caso de Uso.

En los puntos siguientes, se define el significado de cada uno de los campos que componen la plantilla de casos de uso:

- Cabecera: en esta sección se recogen los datos necesarios para la identificación del caso de uso, e información genérica sobre el mismo:
  - Título: texto breve cuyo objetivo es el reconocimiento del caso de uso.
  - Identificador: identificador único del caso de uso. Este identificador está compuesto por las siglas “CU”, seguidas de “-“, y un número de dos cifras, que comienza en “01”, y que se irá incrementando según se vayan añadiendo más casos de uso.
  - Versión: versión actual del caso de uso. El formato de la versión es “X.Y”, donde X representa el número de versión e Y el número de revisión. El número de versión empieza en 1 y se va incrementado en caso de que la especificación del contenido del caso de uso se modifique en gran medida, mientras que el número de revisión comienza en 0, incrementándose en caso de que se produzcan modificaciones de carácter leve en su contenido, o haya errores ortográficos o de formato.
  - Fuente: indica el origen del que procede el caso de uso.
  - Actores: entidades externas al sistema, que interactúan con ella.
  - Objetivo: descripción breve de lo que se pretende conseguir con la ejecución del caso de uso.

- Precondiciones: estado del sistema necesario para la ejecución del caso de uso.
- Postcondiciones: estado del sistema tras la ejecución del caso de uso.
- Escenario básico: escenario en el cual el caso de uso se ejecuta por el camino principal, sin seguir ningún tipo de camino alternativo.
- Escenarios alternativos: conjunto de escenarios que contemplan caminos alternativos al principal, los cuales se llevan a cabo si se cumple una determinada condición en un determinado punto del flujo de ejecución básico.

3.7.3. Casos de uso del actor *Usuario NCTUns*.

En la Ilustración 10 se presenta el diagrama de casos de uso asociado al actor *Usuario NCTUns*.

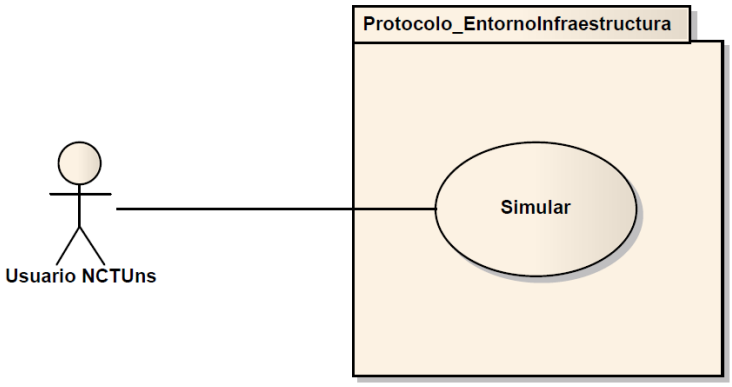

Ilustración 10: diagrama casos de uso *Usuario NCTUns*.

En la Tabla 6, se realiza la especificación del caso de uso *Simular*.

CASO DE USO	
Título: simular.	
Identificador: CU-01	
Versión: 1.0	**Fuente:** cliente.
Actores: *Usuario NCTUns*.	
Objetivo: simular el comportamiento de las distintas entidades que participan en EVIGEN [Bibliografía-1], en un entorno de simulación concreto.	
Precondiciones: ninguna.	
Postcondiciones: • Simulación realizada.	
ESCENARIO BÁSICO	
1. Crear escenario de simulación. 2. Simular. 3. Terminar.	

Tabla 6: Caso de Uso CU-01 simular.

Resulta necesario indicar que el sistema a desarrollar no contempla la implementación de la creación de distintos escenarios de simulación, ni la implementación de la ejecución de las simulaciones con NCTUns

[Referencias-1], ya que dichos aspectos se encuentran ya desarrollados. Sin embargo, resulta necesario especificar este caso de uso de esta forma, ya que para iniciar la ejecución de las distintas entidades, es necesario que el usuario correspondiente interactúe directamente con el simulador.

3.7.4. Casos de uso del actor *Usuario sistema representacion indicadores*.

El diagrama de casos de uso correspondiente con el actor *Usuario sistema representacion indicadores* se muestra en la Ilustración 11.

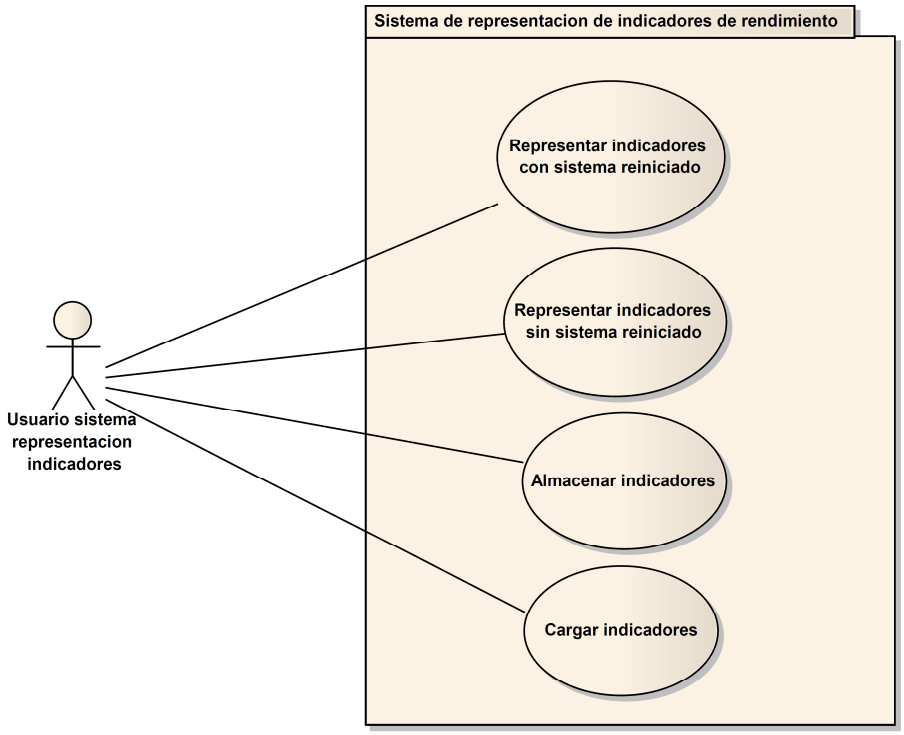

Ilustración 11: diagrama casos de uso 1 Usuario sistema representación de indicadores.

A continuación, se especifican los casos de uso asociados al actor *Usuario sistema representacion indicadores*.

CASO DE USO	
Título: representar indicadores con sistema reiniciado.	
Identificador: CU-02	
Versión: 1.0	**Fuente:** cliente.
Actores: *Usuario sistema representacion indicadores.*	
Objetivo: representar en el sistema de representación de indicadores, los indicadores de rendimiento obtenidos por las distintas entidades, eliminando previamente cualquier información que pudiese estar reflejado en el mismo.	
Precondiciones: ninguna.	
Postcondiciones: • Indicadores de rendimiento representados en el sistema.	
ESCENARIO BÁSICO	
1. Representar indicadores. 2. Terminar.	
ESCENARIO ALTERNATIVOS	
Escenario 1	
1a Sistema de representación previamente funcionando. 1. Parar el sistema. 2. Volver al paso 1.	
Escenario 2	
1b Sistema de representación en estado pausado. 1. Parar el sistema. 2. Volver al paso 1.	

Tabla 7: Caso de Uso CU-02 representar indicadores con sistema reiniciado.

CASO DE USO	
Título: representar indicadores sin sistema reiniciado.	
Identificador: CU-03	
Versión: 1.0	**Fuente:** cliente.
Actores: *Usuario sistema representacion indicadores*.	
Objetivo: representar en el sistema de representación de indicadores, los indicadores de rendimiento obtenidos por las distintas entidades, sin eliminar cualquier información previamente representada en el mismo.	
Precondiciones: ninguna.	
Postcondiciones: • Indicadores de rendimiento representados en el sistema.	
ESCENARIO BÁSICO	
1. Continuar representación indicadores. 2. Terminar.	
ESCENARIO ALTERNATIVOS	
Escenario 1	
1a Sistema de representación previamente funcionando. 1. Terminar.	
Escenario 2	
1b Sistema de representación parado. 1. Representar indicadores. 2. Terminar.	

Tabla 8: Caso de Uso CU-03 representar indicadores sin sistema reiniciado.

CASO DE USO	
Título: almacenar indicadores.	
Identificador: CU-04	
Versión: 1.0	**Fuente:** cliente.
Actores: *Usuario sistema representacion indicadores*.	
Objetivo: almacenar la información correspondiente de los indicadores de rendimiento representados en el sistema en un formato de almacenamiento concreto.	
Precondiciones: • Sistema de representación de indicadores parado. • Indicadores de rendimiento representados en el sistema.	
Postcondiciones: ninguna.	
ESCENARIO BÁSICO	
1. Seleccionar formato de almacenamiento: PDF o Excel 2003. 2. Almacenar la información de los indicadores. 3. Terminar.	
ESCENARIO ALTERNATIVOS	
Escenario 1	
2a Extensión de fichero incorrecta. 1. Confirmar notificación de fallo en la extensión del fichero. 2. Volver al paso 1.	
Escenario 2	
2a Fallo en el almacenamiento. 1. Confirmar notificación de fallo en el almacenamiento. 2. Volver al paso 1.	

Tabla 9: Caso de Uso CU-04 almacenar indicadores.

Resulta imprescindible destacar que para la realización del caso de uso CU-04, es necesario previamente la ejecución del caso de uso CU-02 o el caso de uso CU-03. Por tanto, teniendo en cuenta las relaciones existentes en el modelado de casos de uso, existe una relación de *include* o *extend* entre dichos casos de uso. No obstante, tal y como se especifica en los apuntes sobre modelado de casos de uso de la asignatura de Ingeniería del Software I de la Universidad Carlos III de Madrid [Bibliografía-4], considerando la dificultad presente en distinguir entre ambas, y la recomendación de no hacer uso de ellas a no ser que sea estrictamente necesario, se omite la utilización de dichas relaciones y se especifica como parte de las precondiciones del caso de uso CU-04, las postcondiciones de los casos de uso CU-02 y CU-03.

CASO DE USO	
Título: cargar indicadores.	
Identificador: CU-05	
Versión: 1.0	**Fuente:** cliente.
Actores: *Usuario sistema representacion indicadores.*	
Objetivo: cargar el valor de los indicadores de rendimiento almacenados en un fichero Excel 2003, en el sistema de representación de indicadores.	
Precondiciones: • Sistema de representación de indicadores parado.	
Postcondiciones: • Indicadores de rendimiento representados en el sistema procedentes de fichero Excel.	
ESCENARIO BÁSICO	
1. Seleccionar la ruta del archivo Excel que contiene lo/s indicador/es. 2. Cargar indicadores. 3. Terminar.	
ESCENARIO ALTERNATIVOS	
Escenario 1	
1a Extensión del fichero errónea. 1. Confirmar notificación de fallo en la extensión del fichero. 2. Volver al paso 1.	
Escenario 2	
2a Formato del fichero Excel incorrecto. 1. Confirmar notificación de error en el formato del fichero. 2. Volver al paso 1.	
Escenario 3	
2b Error en la lectura del fichero. 1. Confirmar notificación de error en la lectura del fichero. 2. Volver al paso 1.	

Tabla 10: Caso de Uso CU-05 cargar indicadores de formato Excel.

3.8. Catálogo de requisitos de software.

En este apartado se realiza la especificación de los requisitos de software del sistema, siguiendo como metodología para la organización de los mismos la propuesta por la ESA [Referencias-18].

En primer lugar, se define el formato del catálogo de requisitos de software que es utilizado para su definición, y a continuación, se efectúa la especificación de los requisitos para cada una de las partes en las que se distingue el sistema, extensión de NCTUns [Referencias-1] para la simulación de las entidades de la infraestructura de EVIGEN [Bibliografía-1] e implementación del sistema de representación de indicadores.

3.8.1. Formato para la especificación de requisitos.

El formato del catálogo para la especificación de los requisitos de software es el expuesto en la Tabla 11:

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**

Tabla 11: plantilla catálogo de Requisitos de Software.

A continuación, se describen cada uno de los campos que componen la plantilla del catálogo de requisitos de software:

- Id: clave unívoca asociada al requisito. Este identificador se compone por la sigla "R", seguida de "F" si el requisito es funcional, "I" si es inverso o "NF" si es no funcional. Además, en caso de ser no funcional, se deben adjuntar las siglas correspondientes en función de su tipo:
    - □ Rendimiento: "R".
    - □ Interfaz: "I".
    - □ Operacionales: "O".
    - □ Recursos: "RE".
    - □ Verificación: "V".
    - □ Prueba de aceptación: "PA".
    - □ Documentación: "DO".
    - □ Seguridad: "S".
    - □ Portabilidad: "P".
    - □ Calidad: "C".
    - □ Fiabilidad: "F".
    - □ Mantenibilidad: "M".
    - □ Daño: "DA".

  Después, se debe seguir con "-", y posteriormente, en función de la parte del sistema sobre la que se esté realizando la especificación, se incluye "I" si se trata del sistema de representación de indicadores y "E" en caso contrario. Finalmente, se debe incluir un nuevo guión "-", y a continuación, se especifica un número de dos cifras, el cual comienza en "01", y se va incrementando conforme se especifican nuevos requisitos.
- Versión: versión actual del requisito. El formato de la versión es “X.Y”, donde X representa el número de versión e Y el número de revisión. El número de versión empieza en 1 y se incrementa si se produce una importante modificación en la especificación del requisito, y el número de revisión comienza en 0 y su incremento se realiza si la modificación en su contenido es leve, o contiene errores de formato u ortografía.
- Título: texto breve cuyo objetivo es el reconocimiento unívoco del requisito.
- Descripción: descripción breve del requisito.

- Fuente: indica el origen del cual se ha obtenido el requisito.
- Necesidad: indica el grado de importancia que tiene el cumplimiento de dicho requisito. Los valores que puede tomar esta propiedad son "Alta", "Media" o "Baja".
- Estabilidad: probabilidad de que el requisito cambie durante el desarrollo del Proyecto. Los valores que puede tomar esta propiedad son "Alta", "Media" o "Baja".
- Prioridad: indica la necesidad de que un requisito sea cumplido cuanto antes o no, desde el punto de vista del equipo de desarrollo. Los valores que puede tomar esta propiedad son "Alta", "Media" o "Baja".

En los dos subapartados siguientes se realiza la especificación de los requisitos de software de cada una de las partes del sistema. Además, resulta necesario indicar que en cada una de estas partes, se va a incluir un catálogo por cada tipo de requisito, considerando que no van a ser especificadas las categorías propuestas en la metodología ESA [Referencias-18] que no contengan ningún requisito.

3.8.2. Requisitos de software de la extensión de NCTUns para la simulación del entorno de infraestructura del protocolo EVIGEN.

En este apartado se presentan los requisitos de software correspondientes con la extensión de NCTUns [Referencias-1] para la simulación de las entidades pertenecientes a la infraestructura de EVIGEN [Bibliografía-1].

3.8.2.1. Requisitos funcionales.

En la Tabla 12 se especifican los requisitos funcionales de la extensión de NCTUns [Referencias-1] para la simulación de las entidades de la infraestructura de EVIGEN [Bibliografía-1].

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RF-E-01	1.0	Detección de vehículo infractor.	Detectar una RSU la infracción que ha podido cometer un vehículo que se encuentra circulando próximo a ella, y crear la correspondiente notificación de infracción en caso de producirse.	Cliente.	Alta.	Alta.	Alta.
RF-E-02	1.0	Envío de mensaje con la notificación de la infracción cometida por un vehículo a la DGT.	Enviar un mensaje que contiene la notificación de infracción de un vehículo concreto, desde una RSU a la DGT. Este envío únicamente se realiza si la RSU está conectada directamente con la DGT.	Cliente.	Alta.	Alta.	Alta.
RF-E-03	1.0	Envío de mensaje con la notificación de la infracción cometida por un vehículo a las RSU.	Enviar un mensaje que contiene la notificación de infracción de un determinado vehículo, desde una RSU a sus RSU vecinas. Este envío únicamente se realiza si la RSU no se encuentra directamente conectada con la DGT.	Cliente.	Alta.	Alta.	Alta.
RF-E-04	1.0	Reenvío de mensaje con una notificación de infracción a la DGT.	Reenviar desde una RSU a la DGT, el mensaje con la notificación de la infracción cometida por cierto vehículo procedente de otra RSU. Este envío únicamente se realiza si la RSU se encuentra directamente conectada con la DGT.	Cliente.	Alta.	Alta.	Alta.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RF-E-05	1.0	Reenvío de mensaje con una notificación de infracción a las RSU.	Reenviar desde una RSU a las RSU vecinas a la misma, el mensaje con la notificación de infracción de un vehículo determinado procedente de otra RSU. Esta transmisión se produce en caso de que la RSU no se encuentra directamente conectada con la DGT.	Cliente.	Alta.	Alta.	Alta.
RF-E-06	1.0	Envío de mensaje con una notificación de sanción al vehículo infractor.	Enviar un mensaje que contiene una notificación de sanción, desde una RSU al vehículo infractor al que va destinada. Este envío se realiza únicamente si el vehículo infractor se encuentra próximo a la RSU por una determinada distancia.	Cliente.	Alta.	Alta.	Alta.
RF-E-07	1.0	Envío de mensaje con una notificación de sanción, de una RSU al resto de RSU vecinas.	Enviar un mensaje con la notificación de sanción de un determinado vehículo infractor, desde una RSU a cada RSU vecina a la misma. Este envío se efectúa si el vehículo infractor no se encuentra cerca de la RSU por una distancia concreta.	Cliente.	Alta.	Alta.	Alta.
RF-E-08	1.0	Envío de mensaje con la CRL, de la RSU al resto de RSU vecinas.	Enviar un mensaje que contiene la CRL, desde una RSU al resto de RSU vecinas a la misma.	Cliente.	Media.	Alta.	Media.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RF-E-09	1.0	Envío de mensaje con la CRL, de la RSU a los vehículos próximos a la misma.	Enviar un mensaje con la CRL, desde una RSU al conjunto de vehículos próximos a la misma por una determinada distancia.	Cliente.	Media.	Alta.	Media.
RF-E-10	1.0	Recepción de mensaje con la evidencia de respuesta a una notificación de sanción, desde el vehículo infractor.	Recibir en una RSU procedente del vehículo infractor, un mensaje con una evidencia de respuesta a una notificación de sanción determinada.	Cliente.	Alta.	Alta.	Alta.
RF-E-11	1.0	Envío a la DGT de mensaje con la evidencia de respuesta a una notificación de sanción.	Enviar un mensaje que contiene una evidencia que comprueba la inocencia de un vehículo infractor ante la sanción que se le ha impuesto, desde una RSU a la DGT. Esto es posible si la RSU se encuentra directamente conectada con la DGT.	Cliente.	Alta.	Alta.	Alta.
RF-E-12	1.0	Envío a las RSU de mensaje con la evidencia de respuesta a la notificación de sanción.	Enviar un mensaje con una evidencia utilizada para recurrir una sanción concreta, desde una RSU al resto de RSU vecinas a la misma. Esto ocurre si la DGT y RSU no pueden comunicarse de forma directa.	Cliente.	Alta.	Alta.	Alta.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RF-E-13	1.0	Envío de *beacon* de la RSU a los vehículos próximos a ella.	Enviar, cada determinado tiempo de manera periódica, un *beacon* desde la RSU a los vehículos cercanos a la misma informándoles de su estado.	Cliente.	Alta.	Alta.	Alta.
RF-E-14	1.0	Recepción de mensaje con un testimonio de un vehículo.	Recibir una RSU procedente de un vehículo cercano a la misma, un mensaje que contiene un testimonio dirigido a cierto vehículo infractor.	Cliente.	Media.	Media.	Media.
RF-E-15	1.0	Envío de mensaje con un testimonio al vehículo infractor.	Enviar desde una RSU al vehículo infractor encargado de la creación de la evidencia, un mensaje que contiene un testimonio procedente de un determinado vehículo u otra RSU.	Cliente.	Media.	Media.	Media.
RF-E-16	1.0	Envío de mensaje con un testimonio a vehículos no equipados.	Enviar desde una RSU al conjunto de vehículos no equipados próximos a ella según una cierta distancia, un testimonio procedente de otra RSU o de un vehículo. Este caso se produce si el vehículo infractor al que va dirigido el testimonio no se encuentra cerca de la RSU en ese instante.	Cliente.	Media.	Media.	Media.
RF-E-17	1.0	Envío de mensaje con un testimonio a RSU vecinas.	Enviar desde una RSU a aquellas RSU vecinas a la misma, un testimonio obtenido desde un vehículo u otra RSU. Esto se realiza cuando el vehículo infractor al que va destinado el testimonio no se encuentra cerca de la RSU en ese momento.	Cliente.	Media.	Media.	Media.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RF-E-18	1.0	Creación de notificación de sanción.	Crear una notificación de sanción correspondiente con la notificación de infracción de un determinado vehículo. La DGT es la encargada de crear la notificación de sanción	Cliente.	Alta.	Alta.	Alta.
RF-E-19	1.0	Envío la DGT de mensaje con una notificación de sanción.	Transmitir un mensaje que contiene una notificación de sanción, desde la DGT a las RSU que se encuentran directamente conectadas a la entidad.	Cliente.	Alta.	Alta.	Alta.
RF-E-20	1.0	Reenvío la DGT de mensaje con una notificación de sanción.	Reenviar un mensaje que contiene una notificación de sanción previamente transmitida, desde la DGT a las RSU conectadas a ella, cada cierto tiempo, y un número determinado de veces, mientras no se reciba una evidencia que recurra a la misma.	Cliente.	Media.	Media.	Media.
RF-E-21	1.0	Comprobación de la evidencia.	Verificar la DGT que el dato de consenso recibido en la evidencia es correcto y que se corresponde con los diferentes testimonios aportados por los testigos.	Cliente.	Alta.	Alta.	Alta.
RF-E-22	1.0	Finalización exitosa del protocolo.	Indicar que el protocolo ha finalizado exitosamente si se recibe una evidencia que responda a la notificación de sanción dentro de un intervalo de tiempo concreto.	Cliente.	Alta.	Alta.	Baja.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RF-E-23	1.0	Finalización fallida del protocolo.	Señalar que el protocolo ha finalizado de forma errónea si no se recibe una evidencia que responda a la notificación de sanción en un intervalo de tiempo determinado.	Cliente.	Alta.	Alta.	Baja.
RF-E-24	1.0	Medición del tiempo de respuesta al mensaje de notificación de sanción.	Medir el número de milisegundos desde que se transmite el mensaje que contiene la notificación de sanción, hasta que se recibe el mensaje con la evidencia que responde a dicha notificación.	Cliente.	Alta.	Alta.	Baja.
RF-E-25	1.0	Envío de mensaje de revocación del certificado de un vehículo.	Transmitir un mensaje con el identificador del vehículo cuyo certificado se quiere revocar, desde la DGT a la AC. Este envío se realiza en caso de que la verificación de la evidencia sea incorrecta, o se haya superado el número de reenvíos permitidos de la notificación de sanción sin haber recibido una evidencia que la responda.	Cliente.	Media.	Alta.	Media.
RF-E-26	1.0	Inserción de un nuevo identificador de vehículo en la CRL.	Insertar el identificador del vehículo cuyo certificado la DGT desea revocar en la CRL. Esta inserción se realiza por la AC.	Cliente.	Media.	Alta.	Media.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RF-E-27	1.0	Envío de mensaje con CRL.	Enviar un mensaje con la CRL, desde la AC a las RSU conectadas a la misma, cada vez que se inserte un nuevo identificador de vehículo infractor en la misma.	Cliente.	Media.	Alta.	Media.
RF-E-28	1.0	Reenvío de mensaje con CRL.	Reenviar un mensaje con la CRL, desde la AC a las RSU conectadas a ella, cada vez que transcurra un intervalo de tiempo determinado.	Cliente.	Media.	Alta.	Media.
RF-E-29	1.0	Medición del tamaño de los mensajes.	Medir el número de bytes transmitidos en los mensajes por las distintas entidades, con excepción de los *beacon* de las RSU.	Cliente.	Alta.	Alta.	Baja.
RF-E-30	1.0	Medición del tiempo de envío de los mensajes.	Medir el número de milisegundos empleado por las entidades en la transmisión de los distintos mensajes, con excepción de los *beacon* de las RSU.	Cliente.	Alta.	Alta.	Baja.
RF-E-31	1.0	Medición del tiempo de computación de las operaciones criptográficas.	Medir la cantidad de milisegundos empleados en la computación de las operaciones criptográficas efectuadas durante la ejecución de las entidades.	Cliente.	Alta.	Alta.	Baja.
RF-E-32	1.0	Almacenamiento en log de operaciones criptográficas.	Almacenar en un log las distintas operaciones criptográficas que se realicen durante la ejecución de las entidades.	Cliente.	Media.	Media.	Baja.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RF-E-33	1.0	Envío de información de mensajes de RSU al sistema de visualización de mensajes.	Enviar desde una RSU al sistema de visualización de mensajes, la información de los mensajes del protocolo que intercambia con los vehículos, así como el contenido del *beacon*.	Cliente.	Media.	Alta.	Baja.

Tabla 12: catálogo de Requisitos de Software funcionales de la extensión NCTUns para simulación de entidades de infraestructura de EVIGEN.

3.8.2.2. Requisitos inversos.

En la Tabla 13 se especifican los requisitos inversos de la extensión de NCTUns [Referencias-1] para la simulación de las entidades de la infraestructura de EVIGEN [Bibliografía-1].

Es necesario indicar que estos requisitos no presentan ni prioridad ni necesidad.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RI-E-01	1.0	No sancionar a un mismo vehículo durante un intervalo de tiempo.	No se debe enviar una notificación de sanción a un vehículo que haya sido sancionado previamente dentro de un intervalo de tiempo, si no se ha recibido la evidencia de respuesta a la primera notificación.	Cliente.	-	Media.	-
RI-E-02	1.0	No detectar infracción de un mismo vehículo durante un intervalo de tiempo.	Una RSU no debe enviar una notificación informando de la infracción de un vehículo, si ha enviado una notificación de infracción previa para dicho vehículo dentro de un intervalo de tiempo determinado.	Cliente.	-	Media.	-

Tabla 13: catálogo de Requisitos de Software inversos de la extensión NCTUns para simulación de entidades de infraestructura de EVIGEN.

3.8.2.3. Requisitos no funcionales.

En los subapartados siguientes, se efectúa la especificación de los requisitos no funcionales de la extensión de NCTUns [Referencias-1] para la simulación del entorno de infraestructura de EVIGEN [Bibliografía-1].

3.8.2.3.1. Requisitos de interfaz.

En la Tabla 14 se especifican los requisitos no funcionales de interfaz de la extensión de NCTUns [Referencias-1] para la simulación de las entidades de la infraestructura de EVIGEN [Bibliografía-1].

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RNFI-E-01	1.0	Comunicación con NCTUns [Referencias-1].	El sistema debe integrarse con el simulador NCTUns [Referencias-1], con el fin de que puedan realizarse las diferentes simulaciones del protocolo.	Cliente.	Alta.	Media.	Alta.
RNFI-E-02	1.0	Sistema operativo.	El sistema operativo sobre el que se debe ejecutar el sistema es Fedora 10 [Referencias-2], utilizando el kernel específico para el funcionamiento de NCTUns [Referencias-1].	Ingeniero software.	Alta.	Alta.	Alta.
RNFI-E-03	1.0	Lenguaje de programación.	Para la implementación de las entidades de la infraestructura de EVIGEN, se hace uso del lenguaje C/C++, con el fin de que pueda integrarse con NCTUns [Referencias-1].	Ingeniero software.	Alta.	Media.	Alta.
RNFI-E-04	1.0	Comunicación UDP.	Se debe hacer uso de datagramas UDP para la comunicación entre las distintas entidades del protocolo.	Ingeniero software.	Alta.	Media.	Alta.
RNFI-E-05	1.0	OpenSSL [Referencias-3].	Se debe tener instalado OpenSSL [Referencias-3] para realizar la criptografía.	Ingeniero software.	Alta.	Media.	Media.
RNFI-E-06	1.0	Fichero de log de operaciones criptográficas.	Se debe utilizar un fichero de texto para almacenar el log en el que se describan las distintas operaciones criptográficas.	Ingeniero software.	Media.	Media.	Media.
RNFI-E-07	1.0	Fichero de configuración de entidades.	Se debe hacer uso de un fichero de texto para especificar los parámetros de configuración que necesitan las distintas entidades para su funcionamiento.	Ingeniero software.	Media.	Media.	Media.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RNFI-E-08	1.0	Fichero de configuración para la comunicación con el sistema de representación de indicadores.	Se debe emplear un fichero de texto para incluir los parámetros de configuración que necesitan las entidades para enviar los indicadores de rendimiento al sistema de representación de los mismos.	Ingeniero software.	Media.	Media.	Baja.

Tabla 14: catálogo de Requisitos de Software no funcionales de interfaz de la extensión NCTUns para simulación de entidades de infraestructura de EVIGEN.

3.8.2.3.2. Requisitos operacionales.

En la Tabla 15 se especifican los requisitos no funcionales operacionales de la extensión de NCTUns [Referencias-1] para la simulación de las entidades de la infraestructura de EVIGEN [Bibliografía-1].

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RNFO-E-01	1.0	Formato de entrada del log.	El formato que debe tener cada entrada del log es el siguiente: • Identificador de la entidad que emite la entrada. Elemento opcional. • Fecha y hora en la que se introduce la entrada. • Texto que se desea insertar.	Ingeniero software.	Media.	Media.	Baja.
RNFO-E-02	1.0	Distancia euclídea.	Se debe hacer uso de la distancia euclídea para determinar la distancia entre dos entidades del protocolo.	Ingeniero software.	Media.	Media.	Media.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RNFO-E-03	1.0	Parámetros de configuración de la entidad DGT.	Se le debe proporcionar a la entidad DGT los siguientes parámetros de configuración que necesita para su funcionamiento: • Ruta del certificado de la entidad. • Ruta del fichero en el que se ubica la clave privada. • Ruta del certificado de la autoridad de certificación. • Ruta del log de operaciones criptográficas. • Puerto de escucha de los mensajes de las RSU. • Puerto de envío de mensajes a las RSU. • Puerto de envío de mensajes a la AC. • Ruta en la que se ubica el fichero de configuración para la comunicación con el sistema de representación de indicadores. • Número de saltos máximo que puede efectuar el mensaje con una notificación de sanción.	Ingeniero software.	Media.	Media.	Media.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
			• Cantidad de segundos que deben transcurrir para que la DGT pueda volver a crear una notificación de sanción para un vehículo que ha sido sancionado previamente, si no ha recibido la evidencia que recurra a la correspondiente notificación. • Número máximo de reenvíos que la DGT puede efectuar sobre una misma notificación de sanción.				

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RNFO-E-04	1.0	Parámetros de configuración de la entidad AC.	Se le debe ofrecer a la entidad AC los parámetros de configuración que necesita para su funcionamiento, los cuales se exponen a continuación: • Ruta del certificado de la entidad. • Ruta del fichero en el que se sitúa la clave privada. • Ruta del log de operaciones criptográficas. • Puerto de escucha de los mensajes de la DGT. • Puerto de envío de mensajes a las RSU. • Ruta en la que se sitúa el fichero de configuración para la comunicación con el sistema de representación de indicadores. • Número de saltos máximo que puede realizar el mensaje con la CRL • Número de segundos que deben transcurrir para que se vuelva a reenviar la CRL a las RSU próximas a la misma.	Ingeniero software.	Media.	Media.	Media.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RNFO-E-05	1.0	Parámetros de configuración de la entidad RSU.	Se le debe ofrecer a la entidad RSU los siguientes parámetros de configuración para conseguir su correcto funcionamiento: • Ruta del certificado de la entidad. • Ruta del fichero en el que se ubica la clave privada. • Ruta del certificado de la autoridad de certificación. • Ruta del log de operaciones criptográficas. • Puerto de escucha de los mensajes de la AC. • Puerto de escucha de los mensajes de la DGT. • Puerto de escucha de los mensajes provenientes de los vehículos. • Puerto de escucha de los *beacon* de los vehículos. • Puerto de envío de mensajes a la AC. • Puerto de envío de mensajes a la DGT.	Ingeniero software.	Media.	Media.	Media.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
			• Puerto de envío de mensajes a las RSU. • Puerto de envío de los mensajes con notificaciones de sanción y CRL a los vehículos. • Puerto de envío del *beacon* de las RSU. • Puerto de envío de los mensajes que contienen testimonios a los vehículos. • Ruta en la que se encuentra el fichero de configuración utilizado para la comunicación con el sistema de representación de indicadores. • Distancia máxima para la comunicación entre la RSU y los vehículos. • Cantidad de segundos que deben transcurrir para que la RSU envíe su *beacon* a los vehículos cercanos a la misma.				

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
			• Velocidad máxima permitida, en metros/segundo, a la que puede circular un vehículo sin que se le detecte una infracción por exceso de velocidad. • Porcentaje máximo de error que puede cometer una RSU en la detección de una infracción. • Número de segundos que deben transcurrir para que la RSU pueda volver a enviar a la DGT, una notificación de infracción de un vehículo previamente sancionado.				
RNFO-E-06	1.0	Parámetros de configuración para la comunicación con el sistema de representación de indicadores.	Los parámetros necesarios por las entidades para enviar los indicadores de rendimiento al sistema de representación de los mismos son: • IP de la máquina en la que se ejecuta el sistema de representación de indicadores. • Puerto en el que escucha el sistema de representación de indicadores.	Ingeniero software.	Media.	Media.	Baja.

Tabla 15: catálogo de Requisitos de Software no funcionales operacionales de la extensión NCTUns para simulación de entidades de infraestructura de EVIGEN.

3.8.2.3.3. Requisitos de seguridad.

En la Tabla 16 se especifican los requisitos no funcionales de seguridad de la extensión de NCTUns [Referencias-1] para la simulación de las entidades de la infraestructura de EVIGEN [Bibliografía-1].

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RNFS-01	1.0	Integridad y no repudio.	Se deben firmar los siguientes mensajes, con el fin de asegurar su integridad y no repudio: • Mensaje con la notificación de la infracción cometida por un vehículo, por la RSU que detectó la infracción. • Mensaje de revocación del certificado de un determinado vehículo, por la DGT. • Mensaje con la notificación de sanción, por la DGT. • Mensaje que contiene la CRL, por la AC. Además, en los dos primeros mensajes se debe adjuntar el certificado de la entidad, para permitir la autenticación de los mismos.	Ingeniero software.	Media.	Alta.	Media.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RNFS-02	1.0	Confidencialidad de los mensajes.	Con el fin de mantener la confidencialidad de los mensajes, se realizan las siguientes operaciones: • Descifrado de los tickets que acompañan a los testimonios con la clave privada de la DGT. De esta manera, la DGT es la única que puede acceder al contenido de estos tickets. • Descifrado de las condiciones de los testimonios con la clave simétrica contenida en los tickets descifrados por la DGT. Con esto se consigue que únicamente la DGT pueda observar el contenido de las condiciones de los testimonios.	Ingeniero software.	Media.	Alta.	Media.
RNFS-03	1.0	Ataques de repetición.	Con el objetivo de evitar los ataques de repetición en el protocolo, se incluyen identificadores únicos en los siguientes mensajes: • Mensaje con la notificación de infracción de un vehículo. • Mensaje con una notificación de sanción.	Ingeniero software.	Media.	Alta.	Media.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RNFS-04	1.0	Saturación de la red.	Se añade un tiempo de vida a los siguientes mensajes, con el fin de evitar su circulación indefinida y la saturación de la red: • Mensaje con una notificación de sanción. • Mensaje con la CRL.	Ingeniero software.	Media.	Alta.	Media.

Tabla 16: catálogo de Requisitos de Software no funcionales de seguridad de la extensión NCTUns para simulación de entidades de infraestructura de EVIGEN.

3.8.2.3.4. Requisitos de daño.

En la Tabla 17 se especifican los requisitos no funcionales de daño de la extensión de NCTUns [Referencias-1] para la simulación de las entidades de la infraestructura de EVIGEN [Bibliografía-1].

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RNFDA-01	1.0	Formato de mensajes incorrecto.	Se deben descartar todos los mensajes intercambiados por las entidades si el formato de los mismos es incorrecto.	Ingeniero software.	Alta.	Alta.	Media.
RNFDA-02	1.0	Campos de mensajes incorrectos.	Se deben descartar todos los mensajes transmitidos por las entidades, si los valores de los campos de los mismos son incorrectos.	Ingeniero software.	Alta.	Alta.	Media.
RNFDA-03	1.0	Duplicación en la recepción de mensajes.	Se deben descartar ciertos mensajes que hayan sido recibidos previamente por una entidad, con el fin de evitar duplicados.	Ingeniero software.	Alta.	Alta.	Media.
RNFDA-04	1.0	Expiración del tiempo de vida de los mensajes.	Se deben descartar aquellos mensajes cuyo tiempo de vida haya finalizado.	Ingeniero software.	Media.	Alta.	Media.
RNFDA-05	1.0	Recepción de mensajes de respuesta cuya petición no ha sido enviada.	Se deben descartar aquellos mensajes de respuesta a una petición determinada por una entidad, en caso de que la misma no haya realizado dicha petición.	Ingeniero software.	Media.	Alta.	Media.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RNFDA-06	1.0	Comprobación de certificado incorrecto.	Se deben eliminar los mensajes en los que la comprobación del certificado de la entidad que lo transmite es incorrecta. Esto se efectúa en los casos expuestos a continuación: • Verificación, por parte de la DGT, del certificado de la RSU que detectó la infracción de un determinado vehículo, en el mensaje de notificación de infracción. • Comprobación, por la AC, del certificado de la DGT, en el mensaje de revocación de certificado. • Verificación, por parte de la DGT, del certificado del vehículo infractor en el mensaje con la evidencia de respuesta a la notificación de sanción. • Comprobación, por parte de la DGT, de los certificados de los diferentes testigos que participaron en la creación de la evidencia de respuesta a la sanción.	Ingeniero software.	Media.	Alta.	Media.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RNFDA-07	1.0	Comprobación de firma incorrecta.	Se descartan aquellos mensajes en los que la verificación de la firma realizada por la entidad emisora es errónea. Esto se produce en los casos siguientes: • Comprobación, por la DGT, de la firma de la RSU que detectó la infracción cometida por un vehículo concreto, en el mensaje de notificación de infracción. • Verificación, por parte de la AC, de la firma realizada por la DGT, en el mensaje de revocación de certificado. • Verificación, por la DGT, de la firma del vehículo infractor en el mensaje con la evidencia destinada a recurrir la correspondiente sanción. • Comprobación, por la DGT, de la firma de los distintos testigos que aportaron su testimonio para la creación de la evidencia de respuesta a la notificación de sanción.	Ingeniero software.	Media.	Alta.	Media.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RNFDA-08	1.0	Descifrado asimétrico incorrecto.	Se descartan los mensajes cifrados asimétricamente, si al realizar el descifrado se verifica que éstos no han sido cifrados con la clave pública correspondiente a la entidad receptora del mismo. Esto se aplica en el caso siguiente: • Descifrado de los tickets de los distintos testigos transmitidos en el mensaje con la evidencia de respuesta a una notificación de sanción, en caso de que no hayan sido cifrados con la clave pública de la DGT.	Ingeniero software.	Media.	Alta.	Media.
RNFDA-09	1.0	Descifrado simétrico incorrecto.	Se deben descartar aquellos mensajes transmitidos cifrados simétricamente, en caso de que al realizar el descifrado de los mismos se comprueba que éstos no han sido cifrados con la clave simétrica correspondiente. Esto se produce en el siguiente caso: • Descifrado de las condiciones de los testimonios del mensaje que contiene la evidencia de respuesta a una notificación de sanción, si éstos no han sido cifrados con las claves simétricas adjuntadas en los tickets correspondientes.	Ingeniero software.	Media.	Alta.	Media.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RNFDA-09	1.0	Parámetros de configuración incorrectos.	Se deben proporcionar valores por defecto a los parámetros de configuración de las distintas entidades, en caso de que no se les proporcionen valores, o éstos sean incorrectos. Si no es posible proporcionar un valor por defecto a un determinado parámetro, el programa correspondiente a esa entidad deberá terminar de manera controlada.	Ingeniero software.	Alta.	Alta.	Media.

Tabla 17: catálogo de Requisitos de Software no funcionales de daño de la extensión NCTUns para simulación de entidades de infraestructura de EVIGEN.

3.8.3. Requisitos de software del sistema de representación de indicadores.

En esta apartado se presentan los requisitos de software correspondientes con el sistema de representación de indicadores.

3.8.3.1. Requisitos funcionales.

En la Tabla 18 se especifican los requisitos funcionales del sistema de representación de indicadores.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RF-I-01	1.0	Reinicio del sistema de representación de indicadores de rendimiento.	Reiniciar el sistema de representación de indicadores de rendimiento eliminando toda la información que pudiese estar representada de simulaciones previas.	Cliente.	Alta.	Alta.	Media.
RF-I-02	1.0	Comienzo de representación de indicadores de rendimiento.	Comenzar la representación de indicadores de rendimiento. Previamente a su comienzo, el sistema de representación es reiniciado.	Cliente.	Alta.	Alta.	Alta.
RF-I-03	1.0	Parada de representación de indicadores de rendimiento.	Parar la representación de los indicadores de rendimiento del protocolo. Una vez parado, si se desea seguir con la presentación de indicadores de rendimiento, se debe comenzar de nuevo.	Cliente.	Alta.	Alta.	Alta.
RF-I-04	1.0	Pausa de representación de indicadores de rendimiento.	Pausar la representación de indicadores de rendimiento. En este caso, es posible continuar con la representación sin eliminar los indicadores presentados previamente.	Cliente.	Alta.	Alta.	Alta.
RF-I-05	1.0	Continuación de representación de indicadores de rendimiento.	Continuar con la representación, previamente pausada, de indicadores de rendimiento.	Cliente.	Alta.	Alta.	Alta.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RF-I-06	1.0	Representación de las estadísticas básicas de los indicadores de rendimiento.	Representar, en el sistema de representación de indicadores, las estadísticas básicas que corresponden con la media del tamaño de los mensajes, media del tiempo de envío, media del tiempo de respuesta, media del tiempo de computación, tasas de éxito y fallo en la finalización del protocolo, y tasas de aceptación y rechazo en la participación en el protocolo. También se deben representar el número total de indicadores obtenidos de cada tipo.	Cliente.	Alta.	Media.	Alta.
RF-I-07	1.0	Representación de las estadísticas avanzadas de los indicadores de rendimiento.	Presentar, en el sistema de representación de indicadores, las estadísticas avanzadas compuestas por las estadísticas básicas (ver RF-I-06), gráfica con el tamaño de los mensajes, gráfica con los tiempos de envío, gráfica con los tiempos de respuesta, gráfica con los tiempos de computación, gráfica con las tasas de éxito y fracaso en la finalización del protocolo, y gráfica con las tasas de aceptación y rechazo en la participación en el protocolo.	Cliente.	Alta.	Media.	Alta.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RF-I-08	1.0	Almacenamiento de las estadísticas de los indicadores de rendimiento en PDF.	Almacenar las estadísticas avanzadas del tamaño de los mensajes, tiempo de envío, tiempo de respuesta, tiempo de computación, éxito y fracaso en la finalización del protocolo, y aceptación y rechazo en la participación en el mismo.	Cliente.	Media.	Media.	Media.
RF-I-09	1.0	Almacenamiento de los indicadores de rendimiento en Excel 2003.	Almacenar, en formato Excel 2003, los indicadores correspondientes con el tamaño de los mensajes, tiempo de envío, tiempo de respuesta, tiempo de computación, éxito y fracaso en la finalización del protocolo, y aceptación y rechazo en su participación.	Cliente.	Media.	Media.	Media.
RF-I-10	1.0	Carga de los indicadores de rendimiento de Excel 2003.	Cargar, de un fichero en formato Excel 2003, los indicadores del tamaño de los mensajes, tiempo de envío, tiempo de respuesta, tiempo de computación, éxito y fallo en la finalización del protocolo, y aceptación y rechazo en la participación en el mismo.	Cliente.	Media.	Media.	Media.
RF-I-11	1.0	Configuración del puerto de escucha del sistema de representación de indicadores.	Configurar el puerto por el cual el sistema de representación de indicadores debe recibir las indicadores obtenidos desde las distintas entidades del protocolo. Tras la configuración, el valor del nuevo puerto debe ser almacenado de manera persistente.	Cliente.	Media.	Media.	Media.

Tabla 18: catálogo de Requisitos de Software funcionales del sistema de representación de indicadores.

3.8.3.2. Requisitos no funcionales.

En los subapartados siguientes, se realiza la especificación de los requisitos no funcionales del sistema de representación de indicadores.

3.8.3.2.1. Requisitos de interfaz.

En la Tabla 19 se especifican los requisitos no funcionales de interfaz del sistema de representación de indicadores.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RNFI-I-01	1.0	Lenguaje de programación para el sistema de representación de indicadores.	El lenguaje de programación utilizado para la implementación del sistema de representación de indicadores es Java SE 6.	Ingeniero software.	Alta.	Media.	Alta.
RNFI-I-02	1.0	Comunicación UDP.	Se debe hacer uso de datagramas UDP para la comunicación entre las entidades y el sistema de representación de indicadores.	Ingeniero software.	Alta.	Media.	Alta.
RNFI-I-03	1.0	Java Swing	Se debe hacer uso de Java Swing para la implementación de la interfaz gráfica del sistema de representación de indicadores.	Ingeniero software.	Alta.	Media.	Alta.
RNFI-I-04	1.0	JFreeChart [Referencias-12]	Se debe hacer uso de la librería JFreeChart [Referencias-12] para la creación de los gráficos del sistema de representación de indicadores.	Ingeniero software.	Alta.	Media.	Media.
RNFI-I-05	1.0	iText [Referencias-13]	Se debe utilizar la librería iText [Referencias-13] para el almacenaje de las estadísticas en formato PDF.	Ingeniero software.	Media.	Media.	Media.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RNFI-I-06	1.0	JExcelApi [Referencias-14]	Se debe hacer uso de la librería JExcelApi [Referencias-14] para realizar el almacenaje y carga de los valores de los indicadores de rendimiento en Excel.	Ingeniero software.	Media.	Media.	Media.
RNFI-I-07	1.0	Formato de mensaje en la comunicación entre las entidades del protocolo y el sistema de representación de medidas.	El formato de los mensajes utilizados en la comunicación entre las diferentes entidades participantes del protocolo y el sistema de representación de medidas es NVT ASCII.	Ingeniero software.	Alta.	Media.	Alta.

Tabla 19: catálogo de Requisitos de Software no funcionales de interfaz del sistema de representación de indicadores.

3.8.3.2.2. Requisitos operacionales.

En la Tabla 20 se especifican los requisitos no funcionales operacionales del sistema de representación de indicadores.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RNFO-I-01	1.0	Gráficas de tipo lineal.	Se deben usar gráficas de tipo lineal para la representación de los indicadores del tamaño de los mensajes, tiempo de envío, tiempo de respuesta y tiempo de computación. En el eje x se presenta el tiempo, desde el comienzo de la representación, en el que fue obtenido el indicador, y en el eje y se presenta el valor concreto del indicador.	Ingeniero software.	Alta.	Media.	Alta.
RNFO-I-02	1.0	Gráficas de tipo circular.	Se deben emplear gráficas de tipo circular para la representación de las correspondientes tasas de éxito/fracaso en la finalización del protocolo, y aceptación/rechazo en la participación en el protocolo.	Ingeniero software.	Alta.	Media.	Alta.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RNFO-I-03	1.0	Formato de indicador almacenado en formato Excel 2003.	Almacenar cada indicador en una línea de un fichero Excel 2003. El formato de la entrada es el siguiente: • En la primera celda, se refleja el tiempo en el que fue obtenido el indicador, desde el comienzo de la representación. • En la segunda celda, se almacena el valor del indicador obtenido. En el caso de la representación de las tasas de éxito/fracaso en la finalización del protocolo y aceptación/rechazo en su participación, se emplea la segunda celda para las tasas de éxito y aceptación, y una tercera celda para las tasas de fracaso o rechazo.	Ingeniero software.	Media.	Media.	Media.

Tabla 20: catálogo de Requisitos de Software no funcionales operacionales del sistema de representación de indicadores.

3.8.3.2.3. Requisitos de daño.

En la Tabla 21 se especifican los requisitos no funcionales de daño del sistema de representación de indicadores.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RNFDA-I-01	1.0	Formato de mensajes incorrecto.	Se deben descartar los mensajes enviados al sistema de representación de indicadores, si el formato de los mismos es incorrecto.	Ingeniero software.	Alta.	Alta.	Media.
RNFDA-I-02	1.0	Campos de mensajes incorrectos.	Se deben descartar los mensajes que son enviados al sistema de representación de indicadores, si los valores de los campos de los mismos son incorrectos.	Ingeniero software.	Alta.	Alta.	Media.
RNFDA-I-03	1.0	Mensaje de error en la comunicación del sistema de representación de indicadores.	Se debe mostrar por la interfaz un mensaje de error adecuado informando del mismo, en caso de producirse un fallo en la comunicación del sistema de representación de indicadores.	Ingeniero software.	Media.	Alta.	Media.
RNFDA-I-04	1.0	Mensaje de error en el almacenamiento del sistema de representación de indicadores.	Se debe mostrar por la interfaz un mensaje de error adecuado informando del mismo, en caso de producirse un fallo en la comunicación del sistema de representación de indicadores.	Ingeniero software.	Media.	Alta.	Media.

CATÁLOGO DE REQUISITOS DE SOFTWARE							
Id	**Versión**	**Título**	**Descripción**	**Fuente**	**Necesidad**	**Estabilidad**	**Prioridad**
RNFDA-I-05	1.0	Mensaje de error en la carga del sistema de representación de indicadores.	Se debe mostrar por la interfaz un mensaje informando sobre el error producido en caso de que falle la lectura del fichero Excel 2003 del que se cargan los indicadores de rendimiento.	Ingeniero software.	Media.	Alta.	Media.
RNFDA-I-06	1.0	Mensaje de error si el formato del fichero Excel 2003 es incorrecto.	Se debe presentar por la pantalla un mensaje de error en el que se informe sobre el mismo, en caso de que el formato del fichero Excel 2003 del que se cargan los indicadores es incorrecto.	Ingeniero software.	Media.	Alta.	Media.
RNFDA-I-07	1.0	Mensaje de error en el establecimiento del puerto de escucha del sistema de representación de indicadores.	Se debe mostrar por la interfaz un mensaje de error adecuado en el que se informe sobre el mismo, si el valor configurado del puerto de escucha de los indicadores transmitidos desde las entidades del protocolo es incorrecto.	Ingeniero software.	Media.	Alta.	Media.

Tabla 21: catálogo de Requisitos de Software no funcionales de daño del sistema de representación de indicadores.

3.9. Diseño del plan de pruebas de aceptación.

El objetivo de este apartado es la realización del diseño de las pruebas necesarias para la validación del sistema, en función de los requisitos especificados por el cliente que debe cumplir el mismo. No obstante, con el objetivo de reducir el número de hojas del presente documento, la especificación del diseño de pruebas de aceptación se encuentra en el Anexo I del documento *MemoriaPFC_Anexos.pdf*.

Por tanto, a continuación se presenta la plantilla utilizada para catalogar el diseño de cada una de las pruebas de aceptación.

3.9.1. Formato para la especificación del diseño de las pruebas de aceptación.

En la Tabla 22, se muestra la plantilla utilizada para realizar la especificación del diseño de las pruebas de aceptación:

PRUEBAS DE ACEPTACIÓN				
Id	**Ver.**	**E. prueba**	**Entrada**	**Salida**

Tabla 22: plantilla catálogo Pruebas de Aceptación.

A continuación, se describen cada uno de los campos que componen el catálogo utilizado para la descripción de las pruebas de aceptación:

- Id: identificador asociado a la prueba, cuyo formato es "P-Y-XX", siendo el valor de Y "E" si se realiza el diseño de las pruebas de aceptación de la extensión de NCTUns [Referencias-1] para la simulación del entorno de infraestructura del protocolo, e "I" en caso contrario. XX son dos dígitos que comienzan en 01, y que se van incrementando unitariamente por cada nueva prueba que se diseñe.
- Ver.: versión actual de la prueba de aceptación. El formato de la versión es "X.Y", correspondiéndose X con el número de versión e Y con el número de revisión. El número de versión comienza en 1 y se incrementa en caso de que la especificación de la prueba cambie en gran medida, mientras que el número de revisión empieza en 0 y se incrementa si la especificación de la prueba contiene errores de formato o de ortografía, o el cambio en su contenido es muy leve.
- E. prueba: requisito/s de software que se desea/n verificar con la realización de la prueba.
- Entrada: elementos de entrada necesarios para la ejecución de la prueba.
- Salida: elementos de salida que se esperan obtener tras la ejecución de la prueba, y que suponen la aceptación de la misma.

4. Diseño detallado.

Una vez realizado el análisis del sistema, en esta sección se especifica el diseño detallado del sistema que servirá como guía para el proceso de codificación.

4.1. Diseño del software.

En este apartado se especifica el diseño de clases de cada uno de los componentes del sistema, así como la interacción lógica entre estos componentes para la realización de los casos de uso especificados en el sistema. Este apartado se compone por los siguientes subapartados:

- Diseño detallado de clases: especificación del diagrama de clases, que representen las distintas unidades de implementación, de cada uno de los componentes.
- Diagramas de secuencia: presentación de una serie de diagramas de secuencia entre componentes, de carácter lógico, cuyo objetivo es ilustrar la funcionalidad asociada a los casos de uso sin necesidad de especificar detalles muy concretos.

4.1.1. Diseño detallado de clases.

Como se puede recordar en el análisis, con el fin de facilitar el desarrollo del mismo, se distinguen en el sistema dos partes principales:

- Extensión de NCTUns [Referencias-1] para la simulación del entorno de infraestructura del protocolo EVIGEN [Bibliografía-1].
- Desarrollo del sistema encargado de representar gráficamente los distintos indicadores de rendimiento obtenidos de las entidades del protocolo.

Por tanto, siguiendo el mismo esquema utilizado durante el análisis, se especifican, en primer lugar, los diagramas de clases de los componentes de la extensión de NCTUns [Referencias-1] para la simulación del comportamiento de las entidades pertenecientes a la infraestructura de EVIGEN [Bibliografía-1], y los diagramas de clases de los componentes del sistema de representación de indicadores.

Previamente a la especificación detallada de los componentes, se comentan algunos aspectos que han sido considerados a lo largo del diseño de los mismos, con el fin de simplificar y hacer más clara su descripción:

- Considerando que se van a utilizar distintos hilos de ejecución, se va a hacer uso de *mutex* para controlar los posibles problemas de concurrencia que se puedan producir.
- En determinados componentes, se hace uso de una clase cuyo objetivo es el proporcionar a los clientes de los mismos de información de los posibles errores que se puedan cometer en éstos, ya sean debidos a fallos internos, o a un mal uso por parte de los clientes de los servicios ofrecidos por los mismos. En cada componente, el nombre de la clase presenta la estructura *ErrorX*, donde *X* es un nombre relacionado con la funcionalidad

proporcionada por el componente (por ejemplo, en el componente *Criptografia*, la clase se denomina *ErrorCriptografia*).

4.1.1.1. Diseño detallado de clases de la extensión de NCTUNS para la simulación del entorno de infraestructura del protocolo EVIGEN.

En la Ilustración 12 se presenta el mismo diagrama presentado en el apartado 3.6.1.1, pero señalando el número de subapartado en el que se desarrolla cada uno de los componentes:

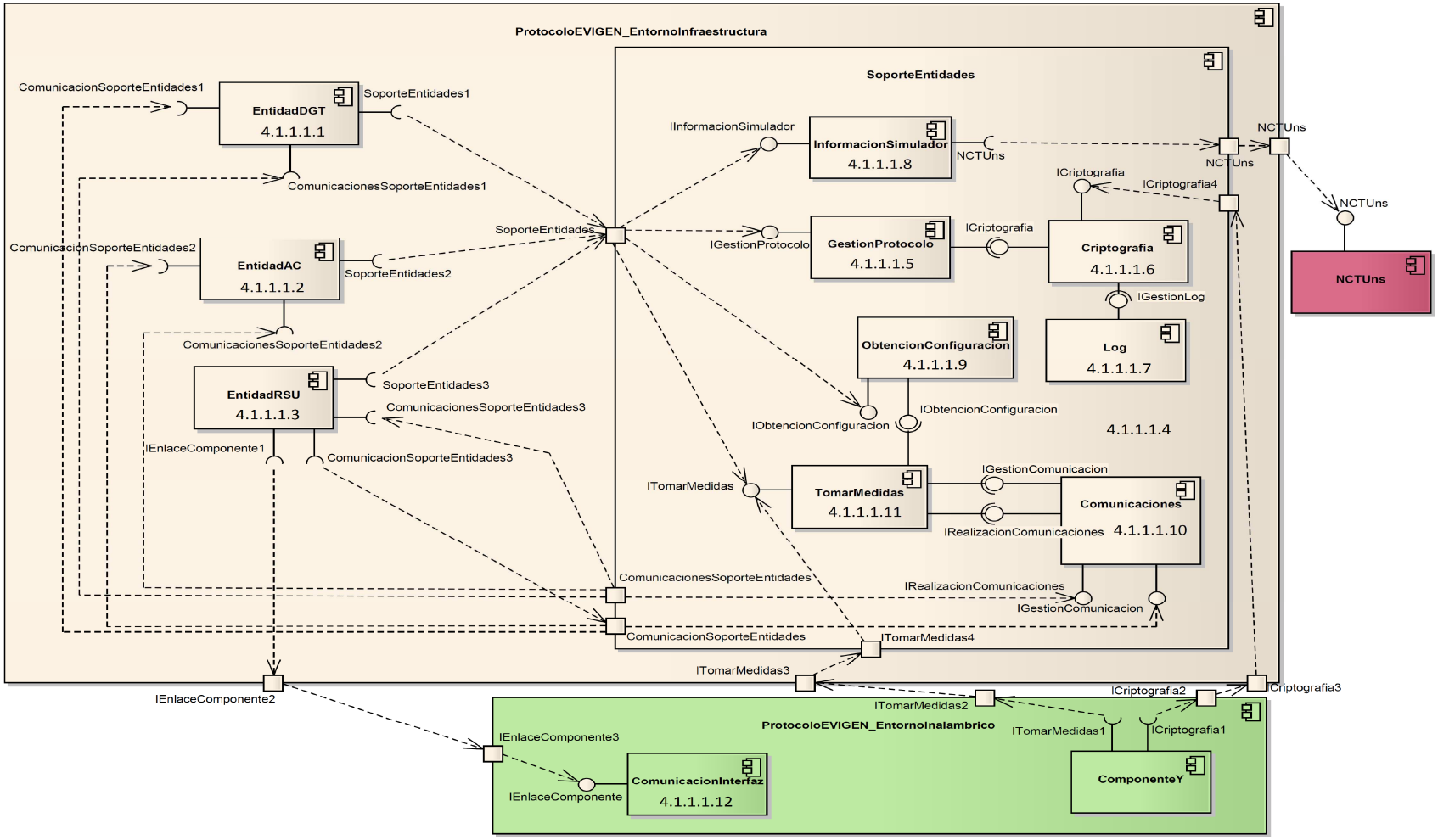

Ilustración 12: esquema de especificación de componentes de la extensión de NCTUns para la simulación de entidades de infraestructura de EVIGEN.

Finalmente, previamente a la descripción de cada componente, resulta necesario describir una limitación que se ha tenido que considerar en el diseño de la solución.

Como se puede recordar del estudio de NCTUns [Referencias-1] realizado en el apartado 3.1, no se pueden conectar de forma física diversas entidades entre sí, debido a que el simulador crearía distintas redes, asignándole a cada entidad una dirección IP distinta por cada nodo al que se encuentre conectada. De esta manera, estas direcciones tendrían que ser especificadas de forma externa al sistema, antes del comienzo de cada simulación. Por ello, todas las entidades deben pertenecer a la misma red, de manera que el simulador le asigne a cada entidad una única dirección IP, que será utilizada por el resto para comunicarse con ella. Por tanto, teniendo en cuenta que todas las entidades deben encontrarse en la misma red, con el fin de aproximarse en la medida de lo posible a la idea original de EVIGEN [Bibliografía-1], en lugar de efectuarse la comunicación entre las entidades que se encuentran conectadas físicamente, ésta se produce entre las entidades cercanas por una distancia de comunicación concreta.

4.1.1.1.1. Diseño del componente EntidadDGT.

En la Ilustración 13 se muestra el diagrama de clases del componente *EntidadDGT*.

En primer lugar, se distingue el módulo *GestionNotificaciones*, cuyo objetivo principal es la creación de notificaciones de sanción a partir de notificaciones de infracción recibidas y de la comprobación de las distintas evidencias de respuesta a notificaciones de sanción. A continuación, se indican las principales operaciones que componen este módulo:

- *comprobarInfraccion*: descartar las notificaciones de infracción de un determinado vehículo, si éste ya ha sido previamente sancionado dentro de un intervalo de tiempo determinado (*segundosEntreNotificaciones*).
- *crearNuevaNotificacion*: crear una notificación de sanción a partir de una determinada notificación de infracción. Cuando se crea una notificación, debe tomar su identificador de *idNotificacion*, el cual comienza en 1 y se va incrementando secuencialmente, y se debe almacenar en *notificaciones* con el objetivo de realizar posteriores reenvíos, hasta que se reciba una evidencia de respuesta a la misma o se supere el número de reenvíos posibles, en cuyo caso se eliminaría de esa lista.
- *comprobarEvidencia*: comprobar si la evidencia recibida se corresponde con los distintos testimonios aportados por los testigos.
- *ejecutarGestionNotificaciones*: operación ejecutada en un hilo de ejecución diferente, encargado de realizar el envío correspondiente de cada notificación de sanción, cada intervalo de tiempo, hasta que se reciba la evidencia de respuesta o se supere el número máximo de reenvíos (*maxReenviosNotificacion*).

Es necesario indicar que para representar las notificaciones que están pendientes de ser respondidas, se debe utilizar la estructura de tipo

NotificacionPendiente, cuyo contenido puede ser elaborado totalmente por el programador.

La clase *DGT* es el núcleo de este componente, ya que se encarga de realizar las operaciones propias de la DGT para efectuar el intercambio de los mensajes correspondientes con el resto de entidades.

Entre estas operaciones se puede encontrar *recibirMensajeCocheInfractor_De_RSU*, la cual se invoca cuando se recibe un mensaje con una notificación de infracción, y se encarga de crear la correspondiente notificación de sanción (*crearNuevaNotificacion* de la clase *GestionNotificaciones*) en caso de ser posible. Además, esta operación comprueba que el mensaje recibido esté libre de errores y que no ha sido recibido previamente, verificando que el identificador del mensaje no se encuentre en la lista de identificadores ya recibidos (*idsUtilizadosMensajeCocheInfractor*). Una vez creada la notificación de sanción, mediante el uso de la operación *enviarNotificacion_A_RSU*, se crea el mensaje correspondiente asignándole como identificador único un número aleatorio que no haya sido usado previamente (el cual se puede comprobar consultando la lista de identificadores ya usados *idsUtilizadosNotificacion*), un tiempo de vida máximo (*TTLMaximo*), y se envía a las RSU más cercanas a la DGT por una distancia de comunicación.

La operación *recibirEvidencia_De_RSU* es invocada cuando se recibe un mensaje con una evidencia destinada a recurrir una sanción correspondiente. Esta operación comprueba que el mensaje recibido es correcto, que no ha sido recibido previamente (consultando el identificador en la lista de identificadores ya utilizados *idsUtilizadosEvidencia*), y que la notificación de sanción asociado a dicha evidencia existe. Una vez hecha estas verificaciones, se debe invocar a la operación *comprobarEvidencia*, y además, se debe indicar que el protocolo ha finalizado correctamente. Por el contrario, la operación *protocoloFinalizadoErroneamente*, se encarga de indicar que la finalización del protocolo ha fracasado.

Como se recordará, esta entidad debe efectuar la medida del tiempo de respuesta a una notificación de sanción. El comienzo de la medición se produce cuando se envía dicha notificación y su finalización cuando se recibe la evidencia de respuesta a la misma. Teniendo en cuenta que se pueden realizar varias mediciones del tiempo de respuesta a la vez, se debe hacer uso de la lista *tiemposRespuestaNotificacion* para almacenar las mismas.

La operación *enviarRevocacionCertificado* se encarga del envío del identificador de un vehículo a la AC para revocar su certificado, en caso de que dicho vehículo haya creado una evidencia cuyo dato de consenso no se corresponda con los testimonios de los testigos, o se haya superado el número máximo de reenvíos de la notificación de sanción que le correspondía. Teniendo en cuenta que al comienzo de su ejecución la DGT no dispone de la IP de la AC, antes de hacer el envío del mensaje de revocación de certificado, debe enviar un mensaje a las RSU solicitándole dicha información

Con el objetivo de que la DGT pueda determinar las RSU que existen y se encuentran cercanas a ella, debe recibir los *beacon* que las RSU envían con su estado. Cuando se recibe este *beacon*, se debe invocar a la operación *recibirBeaconRSU* y dicho *beacon* se almacena en la lista *estadoRSUs*. En un

entorno vehicular real esta comunicación no se produciría, pero teniendo en cuenta que la entidad no dispone de esta información previamente a su ejecución, resulta necesario realizarla.

La lectura del fichero de configuración con los parámetros de la entidad DGT se debe efectuar en la operación *leerConfiguracion*. Resulta necesario añadir que si las rutas de los certificados y clave privada no son especificados en el fichero de configuración, el programa debe acabar de manera controlada. Por otro lado, en caso de no especificarse el resto de parámetros, deben proporcionarse valores por defecto a los mismos. Además, aparte de los parámetros de configuración de la DGT especificados en análisis, se deben incluir en el mismo la distancia máxima de comunicación (*maxDistanciaComunicacion*) entre la DGT y las RSU, y el puerto por el que se reciben los *beacon* de las RSU (*puertoBeaconRSU_DGT*).

La operación *ejecutarDGT* debe enviar de manera periódica un mensaje de estado a las RSU informando de la ubicación de la misma.

El módulo *RecepcionDGT_RSU* se encarga de la recepción de los mensajes provenientes de las RSU, y de invocar a las operaciones correspondientes de la clase *DGT*. Está compuesta por dos hilos que ejecutan las funciones *ejecutarPuertoDGT_RSU*, utilizada para la recepción de los mensajes del protocolo provenientes de las RSU, y *escucharPuertoBeaconRSU_DGT*, para la obtención de los *beacon* de las RSU. De esta manera, se intenta evitar la sobrecarga de la entidad por la llegada masiva de mensajes.

Finalmente, las clases *ComunicacionesDGT* y *ComunicacionDGT* sirven como soporte para la comunicación de la DGT. Encapsulan y delegan, respectivamente, en las clases *ComunicacionesSoporteEntidades* y *ComunicacionSoporteEntidades*, las cuales se describen en el componente *SoporteEntidades*.

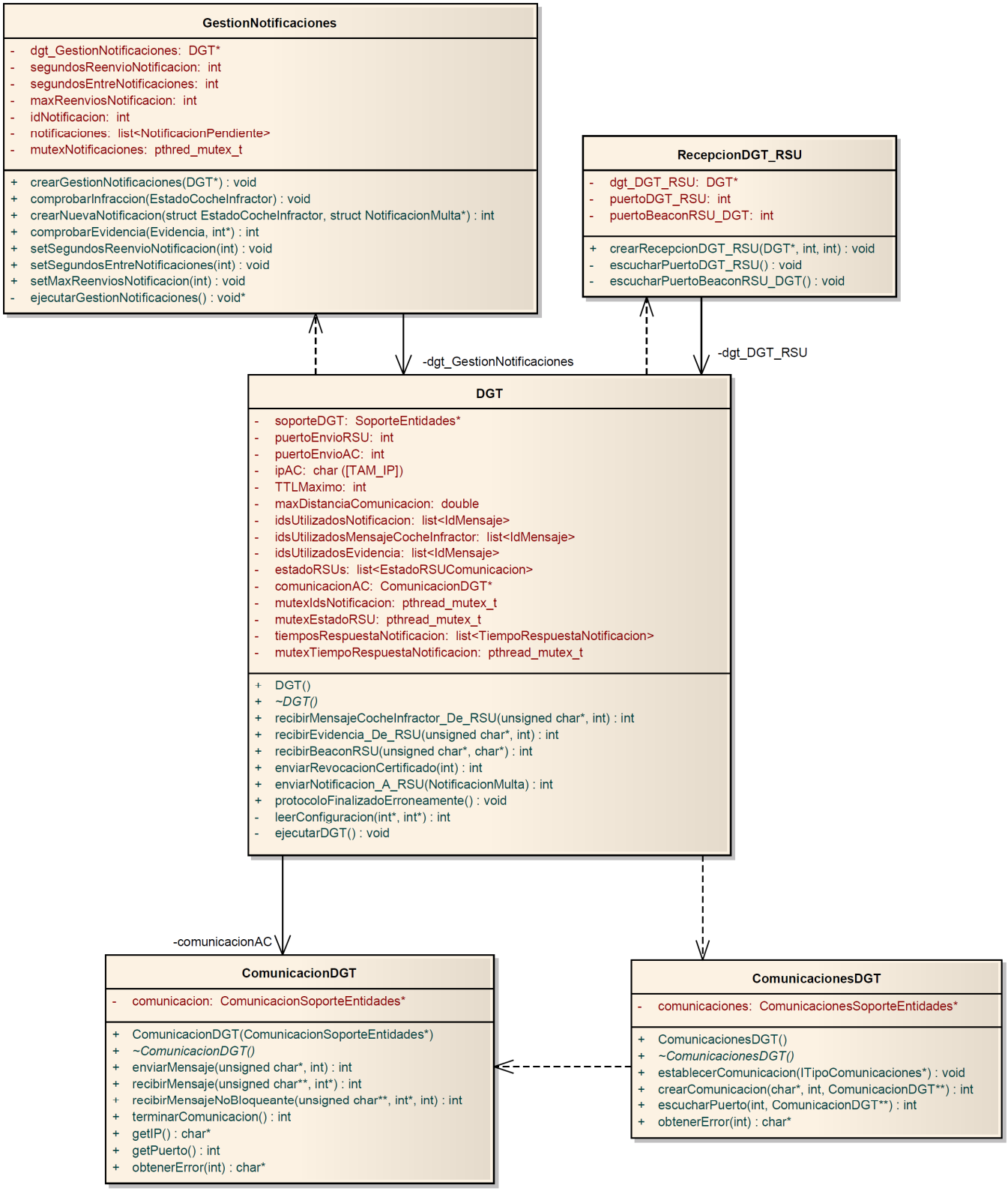

Ilustración 13: diagrama de clases del componente EntidadDGT.

4.1.1.1.2. Diseño del componente EntidadAC.

En la Ilustración 14 se presenta el diseño de clases del componente *EntidadAC*.

En primer lugar, se describe el módulo *GestionCertificados*, cuyo objetivo es el de la gestión de la CRL. En este módulo se destacan las siguientes operaciones:

- *comprobarCoche*: verificar que no se haya insertado el identificador del vehículo en la CRL previamente.
- *revocarCertificado*: insertar en la CRL (*crl*), el identificador de un vehículo cuyo certificado se quiere revocar. Tras la inserción, se invoca a la operación *enviarCRL* para enviarla.
- *ejecutarGestionCertificados*: operación ejecutada en un hilo de ejecución independiente encargado de invocar a *enviarCRL* con el fin de realizar el reenvío de la CRL cada vez que transcurra un intervalo de tiempo determinado (*segundosEnvioCRL*).

Es necesario indicar que en *crl* únicamente se pueden almacenar estructuras de tipo *IdentificadorCoche*, por lo que el programador debe crear dicha estructura para representar el identificador de un vehículo cuyo certificado ha sido revocado.

Posteriormente, se distingue la clase *AC*, que al igual que el componente anterior, se encarga principalmente del envío y recepción de los mensajes con el resto de entidades.

Entre las operaciones que conforman esta clase está *recibirRevocacionCertificado_De_DGT*, en la que se debe recibir el identificador de un vehículo cuyo certificado la DGT quiere revocar.

Otra operación es *enviarCRL_RSU*, en la que la AC debe transmitir un mensaje con la CRL a las RSU más cercanas. A este mensaje se le debe asignar un tiempo de vida máximo (*TTLMaximo*) con el fin de evitar su propagación indefinida por la red.

Al igual que ocurría con el componente anterior, con el fin de determinar la AC a las RSU existentes y más próximas a ella, debe obtener los *beacon* enviados por las RSU, por medio de la operación *recibirBeaconRSU*, y los almacena en la lista *estadoRSUs*.

También se debe enviar de manera periódica un mensaje de estado para informar a las RSU de la presencia de la AC. Esto se efectúa a través de la operación *ejecutarAC*.

Para la lectura de los parámetros de configuración de la entidad AC de un fichero de texto, se debe utilizar la operación *leerConfiguracion*. Al igual que el componente anterior, en caso de que no se especifique la ruta del certificado y fichero con la clave privada, el programa correspondiente con la AC debe finalizar de manera controlada, mientras que al resto de parámetros se le otorga un valor por defecto en caso de que no se especifique su valor en el fichero de configuración. Además, teniendo en cuenta las restricciones impuestas por el diseño, se deben especificar también en este fichero el puerto por el que se

recibe los *beacon* de las RSU (*puertoBeaconRSU_AC*) y la distancia máxima a la que pueden encontrarse las RSU de la AC para poder comunicarse con ellas (*maxDistanciaComunicacion*).

Los módulos *RecepcionAC_DGT* y *RecepcionAC_RSU* deben contener cada uno un hilo encargado de la recepción, respectivamente, de los mensajes de revocación de certificado procedentes de la DGT y de los *beacon* enviados desde las RSU.

Finalmente, del mismo modo que en el componente anterior, las clases *ComunicacionesAC* y *ComunicacionAC* encapsulan la utilización de las clases *ComunicacionesSoporteEntidades* y *ComunicacionSoporteEntidades*, las cuales se especificarán en la descripción del componente *SoporteEntidades*.

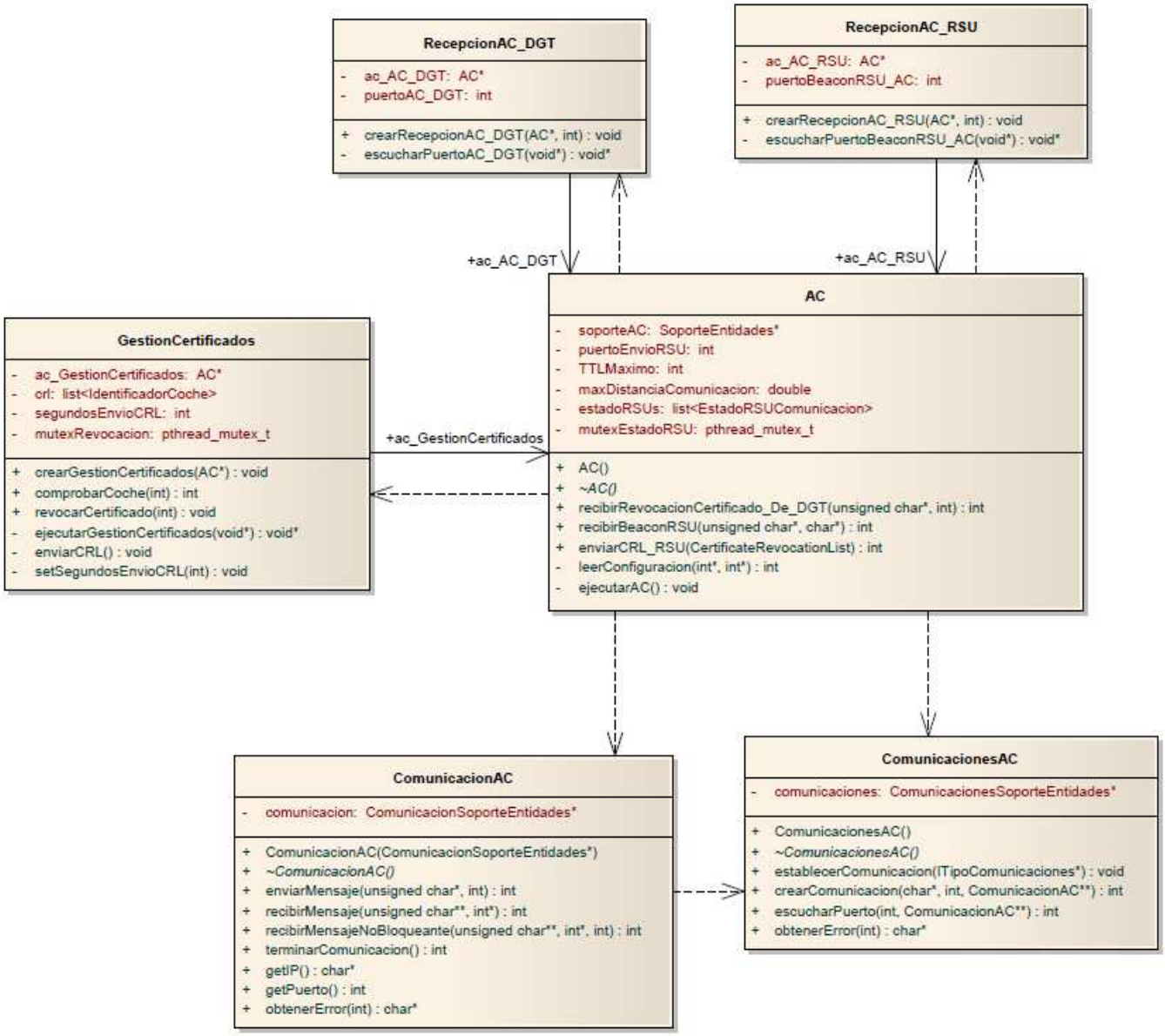

Ilustración 14: diagrama de clases del componente EntidadAC.

4.1.1.1.3. Diseño del componente EntidadRSU.

Como se puede comprobar, teniendo en cuenta que el tamaño de este componente es mayor que los otros, es necesario ilustrar el diagrama de clases en dos ilustraciones distintas.

En la Ilustración 15 se puede distinguir la clase *GestionInfracciones*, cuyo objetivo es determinar si un vehículo ha cometido una determinada infracción, creando la correspondiente notificación en caso de ser así.

La operación *comprobarInfraccion* de esta clase se encarga de realizar esta determinación. Teniendo en cuenta que las RSU creadas no van a disponer de sensores reales que puedan obtener información de los vehículos de su entorno, dicha información va a ser obtenida recibiendo los *beacon* que transmiten los distintos vehículos. Por tanto, cuando una RSU obtenga el *beacon* de un vehículo, debe comprobar que éste se encuentra cercana a ella por una distancia de comunicación determinada, y en caso de ser así, se comprueba si ha cometido una infracción. Como se especificó en el análisis, este sistema se va a limitar a la sanción por exceso de velocidad, por lo que se debe comprobar que la velocidad a la que circula el vehículo es mayor que la velocidad máxima permitida (*velocidadMaxima*). Además, teniendo en cuenta que la velocidad obtenida por las RSU puede diferir de la real, para simular esto, se debe aplicar un porcentaje, obtenido de manera aleatoria, que incremente o decremente la velocidad en dicha proporción. Es necesario indicar también que este porcentaje debe ser menor o igual al porcentaje de fallo en la detección establecido en las RSU (*maximoPorcentajeVariacionVelocidad*).

Otro de los objetivos de la operación *comprobarInfraccion* debe ser la no comprobación de la infracción de un vehículo, si ya ha sido enviada una notificación de infracción del mismo dentro de un intervalo de tiempo determinado (*segundosEntreInfracciones*). Para realizar esto, cuando la RSU crea la notificación de infracción de un vehículo, debe almacenar el identificador del mismo junto con el instante de tiempo en el que fue creada la notificación, en la lista *infraccionesCometidas*, y en caso de que el vehículo vuelva a cometer otra infracción, debe comprobar la diferencia de tiempo entre el instante de tiempo de la nueva infracción y el tiempo almacenado, y en función de esta diferencia, determinar si se puede volver a enviar otra notificación o no. Como se puede observar en la Ilustración 15, a la lista *infraccionesCometidas* únicamente se le pueden añadir estructuras de tipo *InfraccionCometida*, por lo que el programador debe incluir en su contenido el identificador del vehículo sancionado y el instante de tiempo en el que se generó la notificación de infracción.

Al igual que las otras entidades, la clase *RSU* tiene como finalidad la implementación de las operaciones que definen su interacción con el resto de entidades del sistema.

Las primeras operaciones que se encuentran en esta clase son *recibirCRL_De_AC* y *recibirCRL_De_RSU*, las cuales haciendo uso de la operación *recibirCRL* tienen como objetivo recibir el mensaje con la CRL recibido, respectivamente, de la AC o de otra RSU, y de transmitirlo a los vehículos y otras RSU cercanas a ella, mediante el uso de las operaciones *enviarCRL_A_Coches* y *enviarCRL_A_RSU*. La operación *recibirCRL* se debe

encargar también de comprobar que el mensaje recibido sea correcto, que su tiempo de vida no haya expirado, y de decrementar el mismo en una unidad.

Las operaciones *recibirNotificacion_De_DGT* y *recibirNotificacion_De_RSU*, utilizando la operación *recibirNotificacion*, se deben encargar de recibir el mensaje con una notificación de sanción procedentes, de manera respectiva, de la DGT u otra RSU, y de enviarlo al vehículo infractor en caso de que se encuentren circulando cerca de la RSU, o al resto de RSU próximas a ella en caso contrario. La operación *recibirNotificacion* debe comprobar también que el formato y campos del mensaje sean correctos y que el tiempo de vida del mensaje no haya expirado, decrementando el mismo en una unidad. Además, se debe encargar también de verificar que la RSU no haya entregado dicho mensaje previamente al vehículo infractor, comprobando que el identificador del mismo no se encuentre almacenado en la lista de identificadores de este tipo de mensaje ya utilizados (*idsUtilizadosNotificacion*).

La operación *recibirMensajeCocheInfractor_De_RSU* debe encargarse de recibir el mensaje con la notificación de infracción de un vehículo y de reenviarlo a la DGT en caso de que la RSU se encuentra próxima a ésta según la distancia de comunicación, o al resto de RSU cercanas a la misma. Además, debe comprobar que el mensaje recibido no contenga errores y que no haya sido recibido previamente, verificando que el identificador del mismo no esté contenido en la lista de identificadores ya utilizados de este tipo de mensajes (*idsUtilizadosMensajeCocheInfractor*).

Las operaciones *recibirEvidencia_De_Coche* y *recibirEvidencia_De_RSU*, haciendo uso de la operación *recibirEvidencia*, tienen como objetivo recibir el mensaje con una evidencia de respuesta a una notificación de sanción y de transmitirlo a la DGT si la RSU se ubica cercana a ella de acuerdo a una distancia de comunicación, o al resto de RSU próximas a la misma en caso contrario. También se debe encargar de verificar que el mensaje con la evidencia tenga formato y campos correctos, y que no haya sido obtenido anteriormente en la entidad, comprobando que el identificador del mensaje correspondiente no esté insertado en la lista de identificadores de este tipo de mensaje (*idsUtilizadosEvidencia*).

Como se comentó en párrafos anteriores, la determinación de infracción de un vehículo se efectúa a través de la comprobación obtenida de su *beacon*. Este *beacon* se obtiene en la operación *recibirBeaconCoche*, siendo ésta, tras comprobar que el *beacon* es correcto, la encargada de invocar a la operación *comprobarInfraccion* de la clase *GestionInfracciones*. En caso de detectarse una infracción y crearse la correspondiente notificación, esta operación también tiene como objetivo crear el mensaje correspondiente, asignándole un identificador único no contenido en *idsUtilizadosMensajeCocheInfractor*, y de transmitirlo a la DGT si se encuentra próxima a la RSU, o a las RSU cercanas en caso de no ser así. Otro de los objetivos de obtener el *beacon* de los vehículos es para determinar su localización en cada instante, almacenándolo en la lista *estadoCoches*, y de esta manera, determinar si se puede realizar comunicación entre la RSU y éstos.

Con el objetivo de conocer las RSU cercanas a otra según una distancia máxima de comunicación, teniendo en cuenta que no se dispone de esa

información desde un comienzo, la RSU debe obtener los *beacon* enviados por el resto de RSU. Este *beacon* debe ser obtenido en la operación *recibirBeaconRSU*, y se almacena en la lista *estadoRSUs.*

Como se comentó en la descripción del componente *EntidadDGT*, para que la DGT pueda conocer la IP de la AC, le tenía que enviar un mensaje a las RSU solicitándole esta información. Este mensaje lo obtienen las RSU en la operación *recibirPeticionIP_AC*, en el cual se responde a la DGT con la IP de la AC, en caso de que las RSU conozcan dicha información. La forma que deben tener las RSU para conocer la IP de la AC, es recibiendo el mensaje de estado enviado desde esa entidad, el cual se debe obtener en la operación *recibirEstado_De_AC* y almacenar en la variable *estadoAC*.

Teniendo en cuenta que las RSU necesitan conocer la localización de la DGT para determinar su cercanía con ella con el fin de poder comunicarse, deben obtener, mediante el uso de la operación *recibirEstado_De_DGT*, el mensaje de estado enviado desde la misma y almacenarlo en la variable *estadoDGT*.

El *beacon* que la RSU transmite periódicamente cada cierto tiempo (*segundosEnvioBeacon*) a los vehículos cercanos, se realiza en la operación *ejecutarRSU*. Resulta necesario recordar que este *beacon* también lo obtienen la DGT y AC, así como el resto de RSU, para determinar su distancia de comunicación con la RSU.

La última operación a describir en la clase *RSU* es *leerConfiguracion*, que al igual que los componentes anteriores tiene como fin obtener los parámetros de configuración del correspondiente fichero que necesita la RSU para su funcionamiento. En caso de no especificarse el valor de los parámetros en el fichero de configuración, se le deben proporcionar valores por defecto a los mismos, con excepción de las rutas de los certificados de la entidad y AC, y del fichero con la clave privada. Además, se debe utilizar el parámetro de configuración para determinar la distancia máxima a la que pueden estar los vehículos de la RSU para poder comunicarse (*maxDistanciaComunicacion*), para determinar la distancia para la comunicación con el resto de RSU, DGT y AC.

En la Ilustración 15 se puede observar que de la clase *RSU* heredan las clases *RSU_ComportamientoPorDefecto* y *RSU_NotEquipped*. La clase *RSU_NotEquipped*, además de heredar el comportamiento de la clase *RSU*, realiza la funcionalidad propia del rol de los vehículos de tipo no equipado, mientras que la clase *RSU_ComportamientoPorDefecto* realiza la funcionalidad de las RSU sin asumir este rol.

Teniendo en cuenta que los vehículos no equipados tienen como objetivo actuar de intermediarios en el envío de los testimonios de los testigos al vehículo infractor, si la RSU adopta este rol, puede recibir los testimonios del vehículo testigo o vehículos no equipados (operación *recibirTestimonio_De_Coche*), o de otras RSU cercanas a ella que también han adoptado dicho rol (operación *recibirTestimonio_De_RSU*). Ambas operaciones hacen uso de la operación *recibirTestimonio*, que se encarga de comprobar que el mensaje que contiene el testimonio no contenga errores y que su tiempo de vida no haya expirado, decrementando en una unidad el mismo en caso de ser así. Una vez comprobado, utilizando la operación

enviarTestimonio_A_Requester, la RSU envía al vehículo infractor al que va destinado el testimonio, el mensaje con el mismo, si se encuentra circulando próximo a ésta por una distancia de comunicación. En caso contrario, reenvía el mensaje al resto de RSU y vehículos de tipo no equipados próximos a la misma (operación *enviarTestimonio_A_CochesNoEquipados)*.

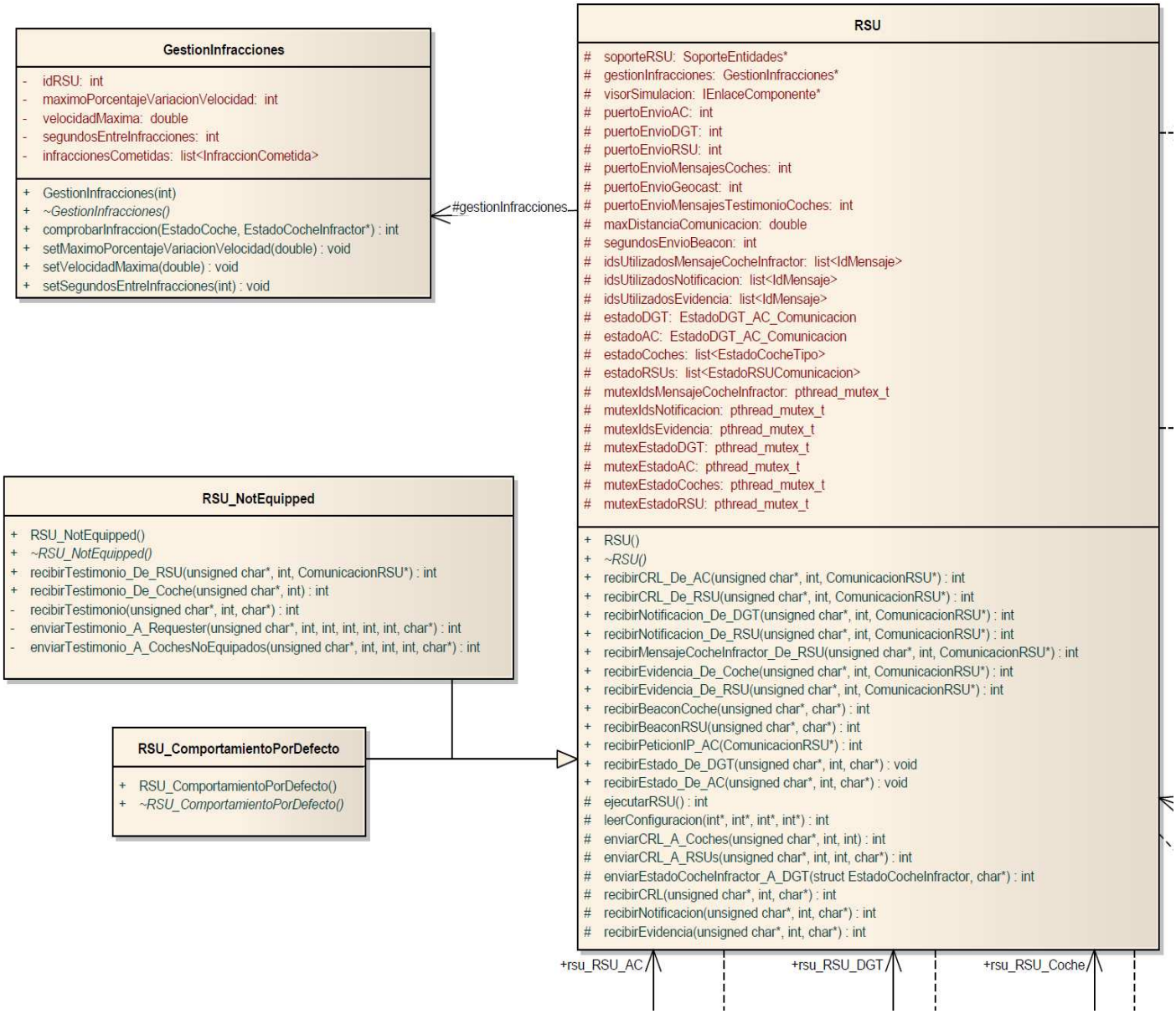

Ilustración 15: diagrama de clases 1 del componente EntidadRSU.

En la Ilustración 16 se observan los módulos *RecepcionRSU_AC*, *RecepcionRSU_DGT*, *RecepcionRSU_RSU* y *RecepcionRSU_Coche*. Los dos primeros deben crear, cada uno de ellos, un hilo de ejecución encargado de recibir los mensajes procedentes de la AC y de la DGT. *RecepcionRSU_RSU* debe crear dos hilos de ejecución con el objetivo de recibir por un lado los mensajes procedentes de otras RSU, y por otro los *beacon* de las RSU, con el fin de evitar la sobrecarga en el envío de mensajes. Por último, el módulo *RecepcionRSU_Coche* debe crear también dos hilos encargados de la recepción de los mensajes y de la recepción de los *beacon* transmitidos desde los vehículos.

Finalmente, de la misma manera que ocurre con los componentes *EntidadDGT* y *EntidadAC*, las clases *ComunicacionesRSU* y *ComunicacionRSU* encapsulan la dependencia con el componente *SoporteEntidades*, y sirve como soporte para la comunicación de las RSU.

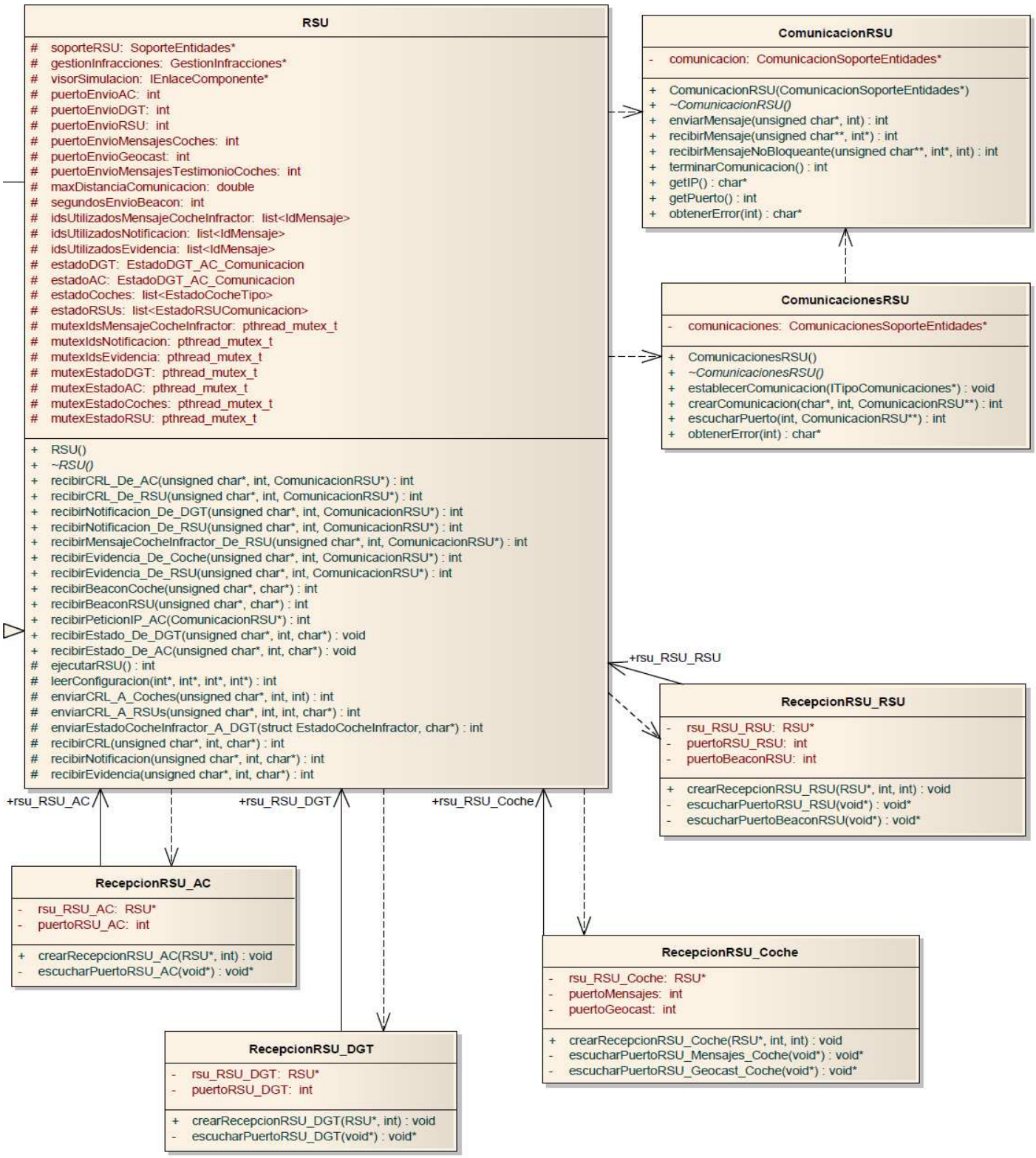

Ilustración 16: diagrama de clases 2 del componente EntidadRSU.

4.1.1.1.4. Diseño del componente SoporteEntidades.

El componente *SoporteEntidades* tiene como objetivo englobar a los distintos componentes que proporcionan distinta funcionalidad de soporte a *EntidadDGT*, *EntidadAC* y *EntidadRSU*.

En la Ilustración 17 se presentan las clases *SoporteEntidades*, *ComunicacionesSoporteEntidades* y *ComunicacionSoporteEntidades*, que implementan los distintos puertos del componente *SoporteEntidades*.

El puerto *SoporteEntidades* tiene como objetivo unificar las interfaces *IInformacionSimulador*, *IGestionProtocolo*, *IObtencionConfiguracion* e *ITomarMedidas*, de manera que las entidades puedan utilizar los servicios ofrecidos por estos componentes, de manera transparente a ellas.

Los puertos *ComunicacionesSoporteEntidades* y *ComunicacionSoporteEntidades* encapsulan, respectivamente, el uso de las interfaces *IRealizacionComunicaciones* e *IGestionComunicacion*, con el objetivo de que las entidades puedan usar sus operaciones sin necesidad de interactuar directamente con el componente interior *Comunicaciones*.

Como ya se especificó en el apartado 3.6, resulta necesario la creación de tres puertos distintos, debido a que el comportamiento del puerto *SoporteEntidades* es incompatible con los puertos *ComunicacionesSoporteEntidades* y *ComunicacionSoporteEntidades.*

Finalmente, teniendo en cuenta que el componente *SoporteEntidades* es la vista que tienen las entidades de los servicios que ofrecen los distintos componentes contenidos en éste, se deben especificar en el mismo una serie de estructuras para representar distintos elementos utilizados en el sistema, cuya definición debe ser común tanto para los componentes *EntidadDGT*, *EntidadRSU* y *EntidadAC*, como para los componentes interiores de *SoporteEntidades*. A continuación, se indican las estructuras que se deben definir, dejando plena libertad al programador para que las cree según mejor lo considere:

- *EstadoCocheInfractor*: representación del contenido de una notificación de infracción.
- *NotificacionMulta*: representación del contenido de una notificación de sanción.
- *CertificateRevocationList*: representación de la lista de identificadores de vehículos cuyo certificado ha sido revocado.
- *RSUEstado*: representación del contenido de los *beacon* de las RSU.
- *EstadoDGT_AC*: representación de la información de estado que envían la DGT o AC a las RSU.
- *EstadoCoche*: representación del contenido de los *beacon* transmitidos desde los vehículos.
- *Evidencia*: representación del contenido de una evidencia de respuesta a una notificación de sanción.
- *TestimonioVehiculo*: representación del contenido de un testimonio aportado por un testigo.

- *IdMensaje*: representación de los identificadores únicos de mensaje, utilizados para evitar los ataques de repetición.

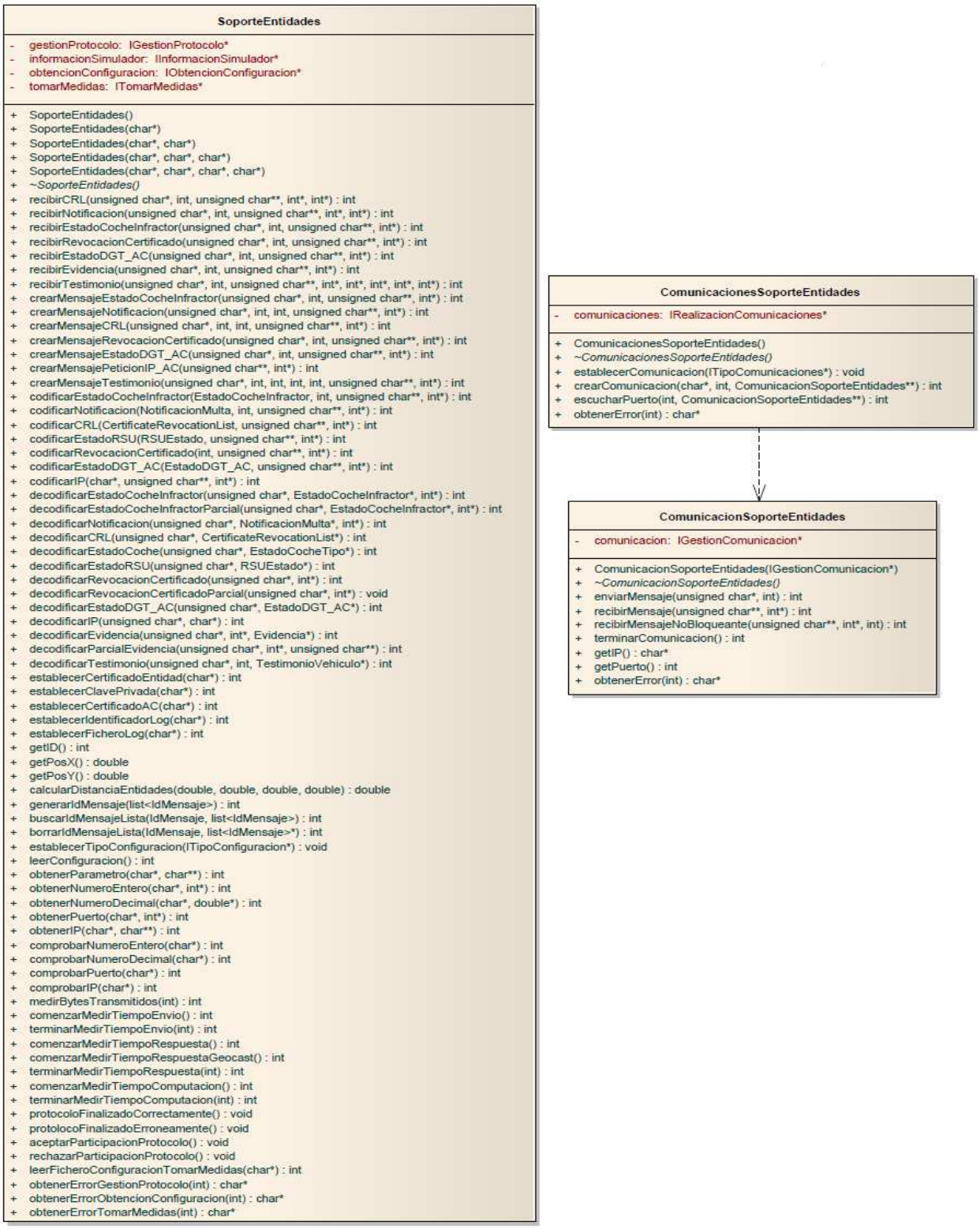

Ilustración 17: diagrama de clases del componente SoporteEntidades.

4.1.1.1.5. Diseño del componente GestionProtocolo.

Como se especificó en la descripción de la arquitectura del sistema del apartado 3.6 de la sección de análisis, el objetivo de este componente es el de

proporcionar funcionalidad de soporte a las entidades para abstraerlas de la realización de ciertas tareas como son la codificación y decodificación de los mensajes, la comprobación de los mensajes, la realización de operaciones criptográficas, etc.

En la Ilustración 18 se presenta la interfaz de este componente *IGestionProtocolo*, y una clase que la implementa *GestionProtocolo*, que sirve de fachada (patrón *Facade* [Bibliografía-3]) para el acceso a la distinta funcionalidad que ofrece el componente.

Ilustración 18: diagrama de clases 1 del componente GestionProtocolo.

Una de las clases a las que debe proporcionar acceso *GestionProtocolo* es *Codificacion*, tal y como se puede observar en la Ilustración 19. El objetivo de esta clase es realizar la codificación y decodificación de los elementos del sistema que son transmitidos de unas entidades a otras, como por ejemplo, las

notificaciones de infracción, las notificaciones de sanción, los *beacon* de las RSU, etc. También se debe encargar de la realización de las distintas operaciones criptográficas que se efectúan durante la ejecución de dichas entidades.

En los puntos siguientes, se especifican las principales operaciones encargadas de la codificación y decodificación de los distintos elementos, indicando para cada una el elemento sobre el que se efectúa la operación, las posibles comprobaciones que se efectúan, y las operaciones criptográficas realizadas.

- *codificarEstadoCocheInfractor*: codificación de notificación de infracción. Ejecuta *comprobacionEstadoCocheInfractor* para realizar las comprobaciones, y efectúa la firma de la RSU.
- *codificarNotificacion*: codificación de notificación de sanción. Ejecuta *comprobacionNotificacion* para realizar las comprobaciones, y efectúa la firma de la DGT.
- *codificarCRL*: codificación de CRL. Ejecuta *comprobacionCRL* para realizar las comprobaciones, y efectúa la firma de la AC.
- *codificarEstadoRSU*: codificación de *beacon* de RSU. Comprueba que el identificador de la RSU sea mayor o igual que 0.
- *codificarRevocacionCertificado*: codificación de identificador de vehículo con certificado revocado. Comprueba que el identificador del vehículo sea mayor o igual que 0, y efectúa la firma de la DGT.
- *decodificarEstadoCocheInfractor*: decodificación de notificación de infracción. Ejecuta *comprobacionEstadoCocheInfractor*, y verifica certificado y firma de RSU.
- *decodificarNotificacion*: decodificación de notificación de sanción. Ejecuta *comprobacionNotificacion.*
- *decodificarCRL*: decodificación de CRL. Ejecuta *comprobacionCRL*.
- *decodificarEstadoCoche*: decodificación de *beacon* de vehículo. Ejecuta *comprobacionEstadoCoche*.
- *decodificarEstadoRSU*: decodificación de *beacon* de RSU. Comprueba que el identificador de la RSU sea mayor o igual que 0.
- *decodificarRevocacionCertificado*: decodificación del identificador de vehículo con certificado revocado. Comprueba que el identificador del vehículo sea mayor o igual que 0, y verifica certificado y firma de la DGT.
- *decodificarEvidencia*: decodificación de evidencia. Efectúa operaciones criptográficas para verificar el mensaje con la evidencia, y efectúa las siguientes comprobaciones:
  - El identificador del mensaje que contiene la evidencia debe ser mayor o igual que 0.
  - El identificador de la notificación de sanción al que responde la evidencia debe ser mayor que 0.

- El dato de consenso alcanzado en base a los testimonios aportados por los testigos debe tener un valor válido. Teniendo en cuenta que la infracción que se va a tratar en el sistema es por exceso de velocidad, comprobar que la velocidad alcanzada como consenso es mayor o igual que 0.
- Por cada condición de testimonio:
  - El extremo menor del intervalo que contiene la velocidad de consenso debe ser mayor o igual que 0, y menor o igual que dicha velocidad.
  - El extremo mayor del intervalo con la velocidad de consenso debe ser mayor o igual que 0, y mayor o igual que esa velocidad.

- *decodificarTestimonio*: decodificación de testimonio. Comprueba que el identificador del vehículo sea mayor o igual que 0.

Además, en esta clase se deben implementar una serie de operaciones encargadas de verificar la corrección de ciertos elementos que se codifican o decodifican.

- *comprobacionEstadoCoche*: comprobar que el *beacon* recibido de los vehículos es correcto. Las comprobaciones que efectúa son las siguientes:
  - El identificador del vehículo debe ser mayor o igual que 0.
  - La velocidad a la que circula el vehículo tiene que ser mayor o igual que 0.
- *comprobacionEstadoCocheInfractor*: verificar que una notificación de infracción es válida. Las comprobaciones que realiza son:
  - El identificador del vehículo infractor debe ser mayor o igual que 0.
  - La velocidad a la que circulaba el vehículo infractor en el momento en el que se le detectó la infracción debe ser mayor o igual que 0.
  - El identificador del procedimiento que se ha infringido debe tener un valor correcto.
  - El identificador de la RSU que detectó la infracción debe ser mayor o igual que 0.
  - El identificador del mensaje con la notificación de infracción debe ser mayor o igual que 0.
- *comprobacionNotificacion*: comprobar que una determinada notificación de sanción no contiene errores. Las verificaciones que efectúa son las siguientes:
  - El identificador del vehículo sancionado debe ser mayor o igual que 0.
  - El identificador del procedimiento de seguridad vial que ha sido infringido debe ser mayor que 0.

- El identificador de la RSU que detectó la infracción debe ser mayor o igual que 0.
- El identificador de la notificación debe ser mayor que 0.
- La información de la infracción debe ser correcta. Recordando que en el sistema a desarrollar la infracción se limita a exceso de velocidad, que la velocidad a la que circulaba el vehículo sea mayor o igual que 0.
- El identificador del mensaje con la notificación debe ser mayor o igual que 0.

♦ comprobacionCRL: verificar que los identificadores de los vehículos contenidos en la CRL son mayores o iguales que 0.

Ilustración 19: diagrama de clases 2 del componente GestionProtocolo.

Descrita la clase *Codificacion*, en la Ilustración 20 se presenta la clase *ComunicacionesGestionProtocolo*, la cual se encarga de crear los mensajes que transmitirán las distintas entidades durante su ejecución. Para ello, a las operaciones que componen esta clase se les debe proporcionar el contenido ya codificado (clase *Codificacion*) a enviar en el mensaje. Además de crear los

mensajes, esta clase también tiene como objetivo la extracción del contenido de los mismos.

Para la extracción de la información de los mensajes se distinguen principalmente las operaciones *recibirCRL*, *recibirNotificación*, *recibirEstadoCocheInfractor*, *recibirRevocacionCertificado*, *recibirEvidencia* y *recibirTestimonio*, encargadas de extraer respectivamente el contenido del mensaje con la CRL, la notificación de sanción, la notificación de infracción, el identificador del vehículo cuyo certificado se desea revocar, la evidencia de respuesta a una notificación de sanción y un testimonio. Además, las operaciones *recibirCRL*, *recibirNotificacion* y *recibirTestimonio* también tienen como propósito verificar que el tiempo de vida del mensaje no haya expirado.

En lo que respecta a las operaciones encargadas de la creación de los mensajes a enviar por las distintas entidades se distinguen principalmente *crearMensajeEstadoCocheInfractor*, *crearMensajeNotificacion*, *crearMensajeCRL*, *crearMensajeRevocacionCertificado* y *crearMensajeTestimonio*, cuyo objetivo es crear respectivamente el mensaje con el contenido de una notificación de infracción, de una notificación de sanción, de la CRL, de un identificador de vehículo cuyo certificado se va a revocar y de un testimonio. Además, en las operaciones *crearMensajeNotificacion*, *crearMensajeCRL* y *crearMensajeTestimonio*, se debe adjuntar el tiempo de vida actual del mensaje, teniendo en cuenta el número de saltos que haya podido realizar previamente.

Otra de las clases que se pueden encontrar en este componente es *FuncionesGestionProtocolo*, la cual se encarga de proporcionar a las entidades de cierta funcionalidad que les sirva de soporte para determinar la distancia con otra entidad, o para realizar la gestión de los identificadores unívocos de los distintos mensajes del protocolo. A continuación, se describen las principales operaciones que componen esta clase:

- *calcularDistanciaEntidades*: calcular la distancia euclídea a la que se encuentran dos entidades, dentro del entorno vehicular simulado.
- *generarIdMensaje*: generar un identificador aleatorio para un mensaje, que no se encuentre contenido en la lista proporcionada como parámetro.
- *buscarIdMensaje*: realizar la determinación de que un identificador de mensaje determinado se encuentre almacenado o no en la lista de identificadores introducida como parámetro.

Finalmente, resulta necesario destacar que, aunque este componente es utilizado únicamente por las entidades DGT, AC y RSU, nuevas entidades que puedan aparecer en el protocolo y que compartan funcionalidad con éstas, podrían utilizar los servicios proporcionados por este componente.

Además, teniendo en cuenta que este componente abstrae de la realización de las operaciones de bajo nivel de codificación y decodificación, modificaciones en estas operaciones no supone la realización de cambios en las entidades.

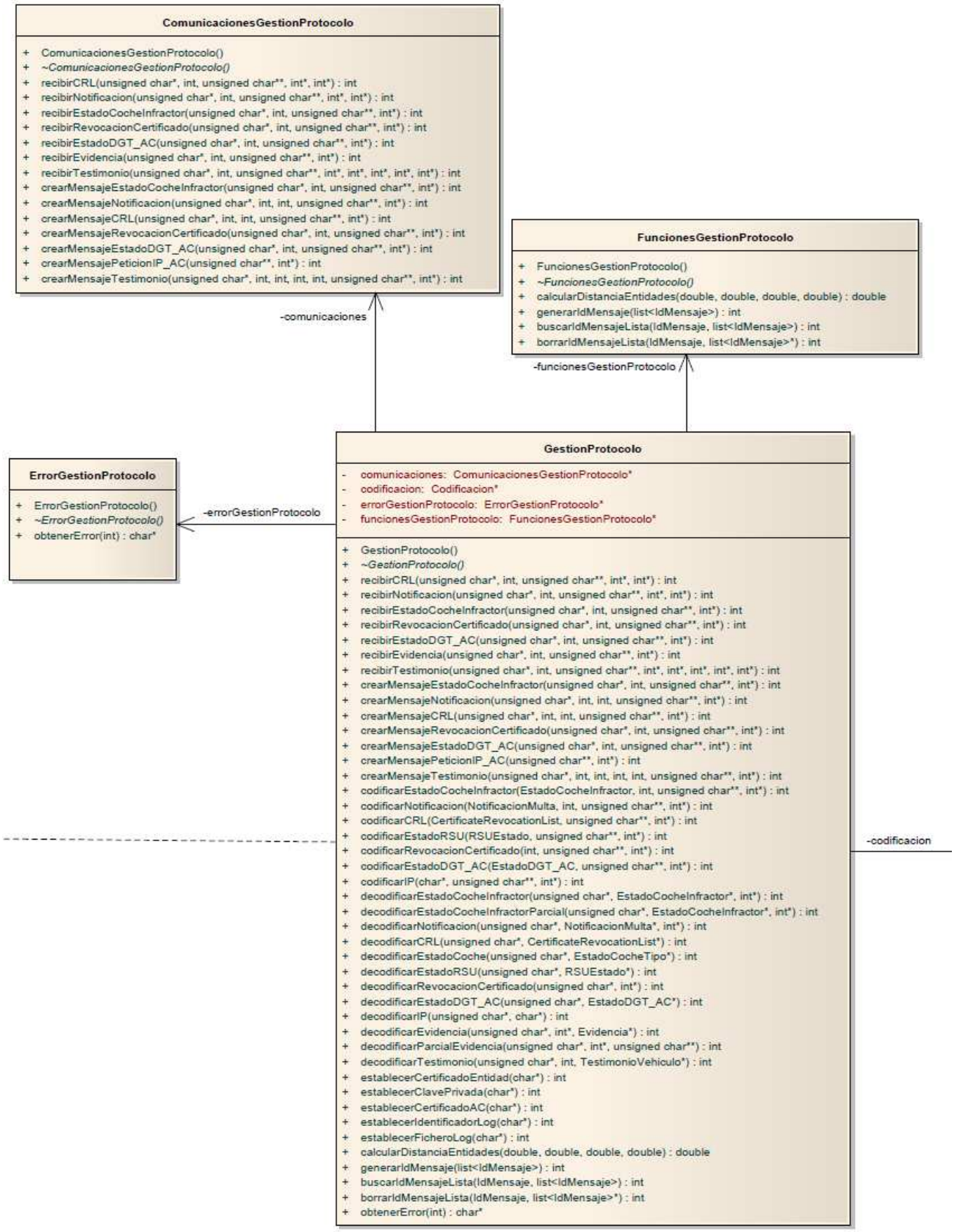

Ilustración 20: diagrama de clases 3 del componente GestionProtocolo.

4.1.1.1.6. Diseño del componente Criptografia.

Como se recordará, el objetivo del componente *Criptografia* es el de proporcionar soporte para la realización de distintas operaciones criptográficas, siendo este componente, por tanto, el utilizado para aplicar la criptografía a los distintos mensajes intercambiados entre las entidades. Como ya fue indicado en el apartado 3.6.1, este componente implementa el módulo, especificado como uno de los objetivos de este Proyecto, encargado de ofrecer distintas operaciones criptográficas a los protocolos que se puedan simular en NCTUns [Referencias-1].

Como se puede comprobar, en la Ilustración 21 se puede observar la interfaz *ICriptografia* proporcionada por este componente para que los clientes hagan uso de sus servicios. También se puede observar la clase *Criptografia*, que implementa a *ICriptografia*, proporcionando una fachada de acceso a la distinta funcionalidad (patrón *Facade* [Bibliografía-3]), repartida entre las distintas clases del componente.

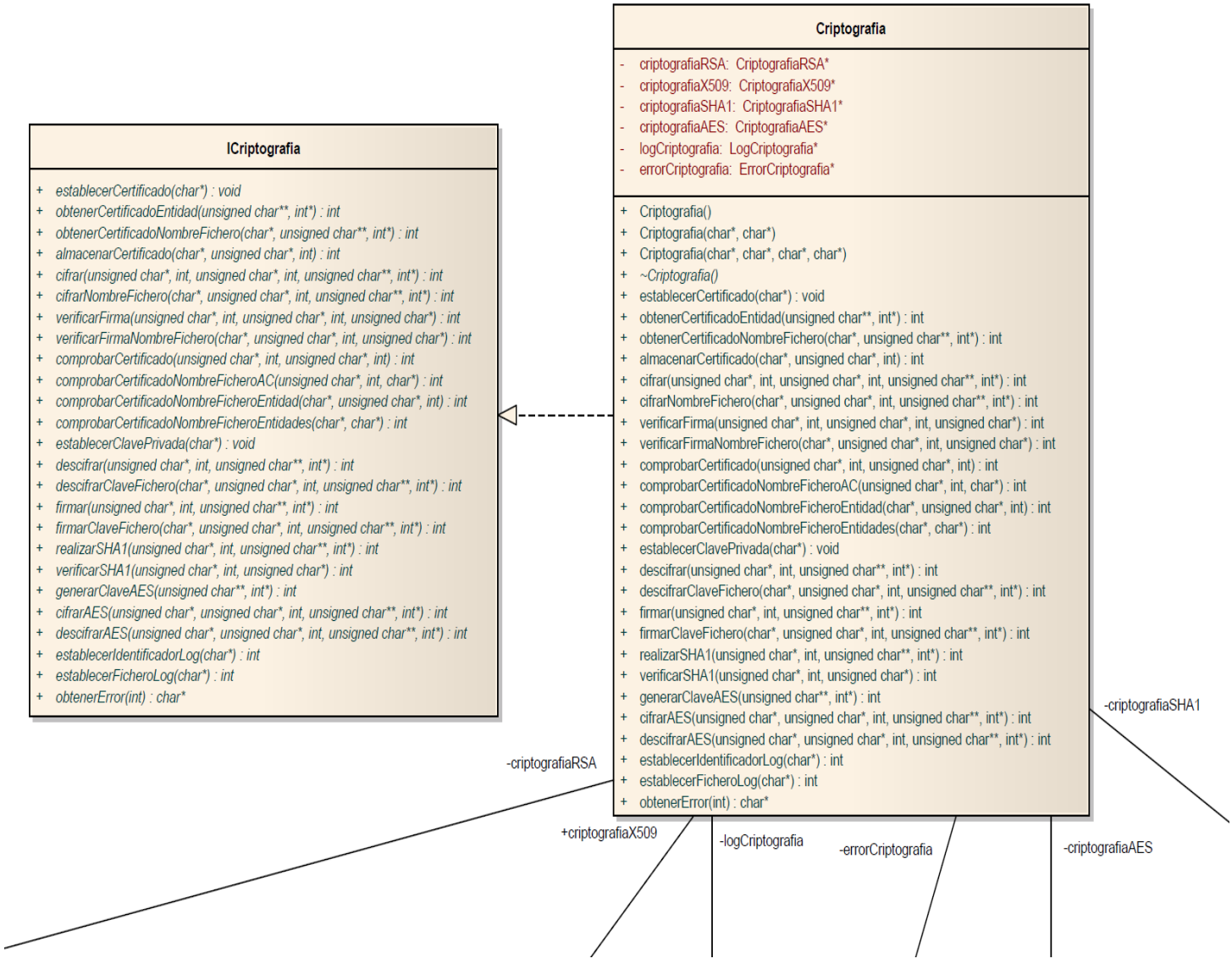

Ilustración 21: diagrama de clases 1 del componente Criptografia.

En la Ilustración 22 se encuentran las clases que implementan la distinta funcionalidad ofrecida por el componente *Criptografia*.

La primera de ellas es *CriptografiaX509*, que se encarga de la realización de las operaciones criptográficas que se pueden efectuar con certificados X509. A continuación, se exponen las principales operaciones que conforman esta clase:

- *establecerCertificado*: indicar la ruta en la que se ubica el certificado de la entidad. De esta manera, una vez se ejecuta esta operación, todas las operaciones criptográficas que impliquen la participación del certificado de la entidad utilizarán éste.
- *obtenerCertificadoEntidad*: obtener el certificado de la entidad codificado, junto con su longitud, habiendo previamente indicado su ruta con la operación *establecerCertificado*.
- *almacenarCertificado*: almacenar el certificado en disco, introducido de manera codificada, en la ruta especificada como parámetro.
- *cifrar:* comprobar el tipo del certificado X509 especificado en la operación *establecerCertificado*, y realizar el cifrado correspondiente según ese tipo. Teniendo en cuenta que en el protocolo a desarrollar el cifrado asimétrico es RSA, se debe invocar a la operación *cifrar* de la clase *CriptografiaRSA* con la clave pública del certificado, ignorando el resto de cifrados si el tipo del certificado es distinto a RSA.
- *verificarFirma*: comprobar el tipo del certificado X509 de la entidad establecido en la operación *establecerCertificado*, y efectuar la verificación de firma correspondiente con ese tipo. En caso de que el tipo sea RSA, se debe invocar a la operación *verificarFirma* de la clase *CriptografiaRSA* con la clave pública del certificado, y en caso contrario, no se realiza la operación.
- *comprobarCertificado*: realizar la comprobación del certificado de una entidad y del certificado de la autoridad de certificación que lo emite, mediante el uso de la operación *comprobarValidezCertificados*. En esta operación, se deben introducir ambos certificados de manera codificada.
- *comprobarValidezCertificados*: realizar las comprobaciones necesarias para validar un certificado. Entre estas comprobaciones se encuentra verificar que el certificado a validar se encuentra firmado por la autoridad de certificación, comprobar que los certificados son válidos y no han expirado, etc.

En esta clase existen otras operaciones como *cifrarNombreFichero*, *verificarFirmaNombreFichero*, que son equivalentes de manera respectiva a *cifrar* y *verificarFirma*, pero especificando el nombre del certificado a utilizar como parámetro.

Es necesario indicar que en esta clase, cada vez que se lea o escriba un certificado de disco, se debe almacenar en la lista *certificados*. De esta manera, si se desea volver a utilizar el mismo certificado en el futuro, se obtiene directamente de memoria, sin necesidad de tener que acceder nuevamente al disco.

Teniendo en cuenta que la lista *certificados* únicamente acepta estructuras de tipo *NodoCertificado*, el programador debe elaborar de la manera que quiera su contenido, de manera que se almacene en el mismo el certificado obtenido o almacenado en disco.

Otra de las clases que aparecen en la Ilustración 22 es *CriptografiaRSA*, cuyo objetivo es el desarrollo de operaciones criptográficas haciendo uso del algoritmo RSA. En los puntos siguientes, se describen las operaciones principales que componen esta clase:

- *establecerClavePrivada*: señalar la ruta en la que se encuentra el fichero que contiene la clave privada de una entidad. Especificada esta clave privada, el resto de operaciones de descifrado y firma RSA se deben realizar con la misma.
- *descifrar*: realizar el descifrado de un conjunto de datos cifrados por medio del algoritmo RSA. Este descifrado se debe realizar utilizando la clave privada especificada en *establecerClavePrivada*.
- *firmar*: realizar la firma con RSA de un determinado conjunto de datos, utilizando la clave privada especificada en *establecerClavePrivada*.
- *cifrar*: realizar el cifrado de un conjunto de datos, haciendo uso del algoritmo RSA, con la clave pública especificada como parámetro.
- *verificarFirma*: realizar la verificación de una firma concreta, utilizando la clave pública RSA indicada como parámetro.

Al igual que en la clase *CriptografiaX509*, existen otras operaciones como *descifrarClaveFichero*, la cual es equivalente a *descifrar* pero especificando el fichero con la clave privada como parámetro.

Empleando la lógica utilizada en la clase *CriptografiaX509*, con el objetivo de reducir los accesos a discos, cada vez que se lea una clave privada de fichero, se debe almacenar en memoria, en la lista *clavesPrivadas*.

Con el objetivo de insertar las claves privadas en la lista *clavesPrivadas*, el programador debe desarrollar el contenido de la estructura *NodoClavePrivada* con plena libertad, de manera que se almacene en dicha estructura la clave privada obtenida de disco.

La clase *CriptografiaSHA1* se encarga de la gestión de las operaciones para la realización (*realizacionSHA1*) y verificación (*verificarSHA1*) de los resúmenes SHA1.

Como se puede comprobar en la Ilustración 22, otra de las clases que componen este componente es *CriptografiaAES*, cuyo objetivo es la realización de las operaciones criptográficas de cifrado (*cifrarAES*) y descifrado *(descifrarAES*), propias del algoritmo AES. También ofrece una operación para la generación de clave AES, *generarClaveAES*.

También se encuentra la clase *LogCriptografia*, que tiene como misión ofrecer al resto de clases de operaciones para realizar el almacenaje de las operaciones criptográficas en el log. Las operaciones más destacables de esta clase son *escribirEnLog*, encargada de almacenar un texto determinado en una entrada del log, y *escribirEnLogMensajeError*, cuyo objetivo es almacenar en el log un mensaje de error concreto, obtenido de la clase *ErrorCriptografia*.

Finalmente, es necesario destacar que este componente, aunque es utilizado para realizar las operaciones criptográficas sobre los distintos mensajes del protocolo, puede ser utilizado para proporcionar criptografía a cualquier aplicación desarrollada en C/C++.

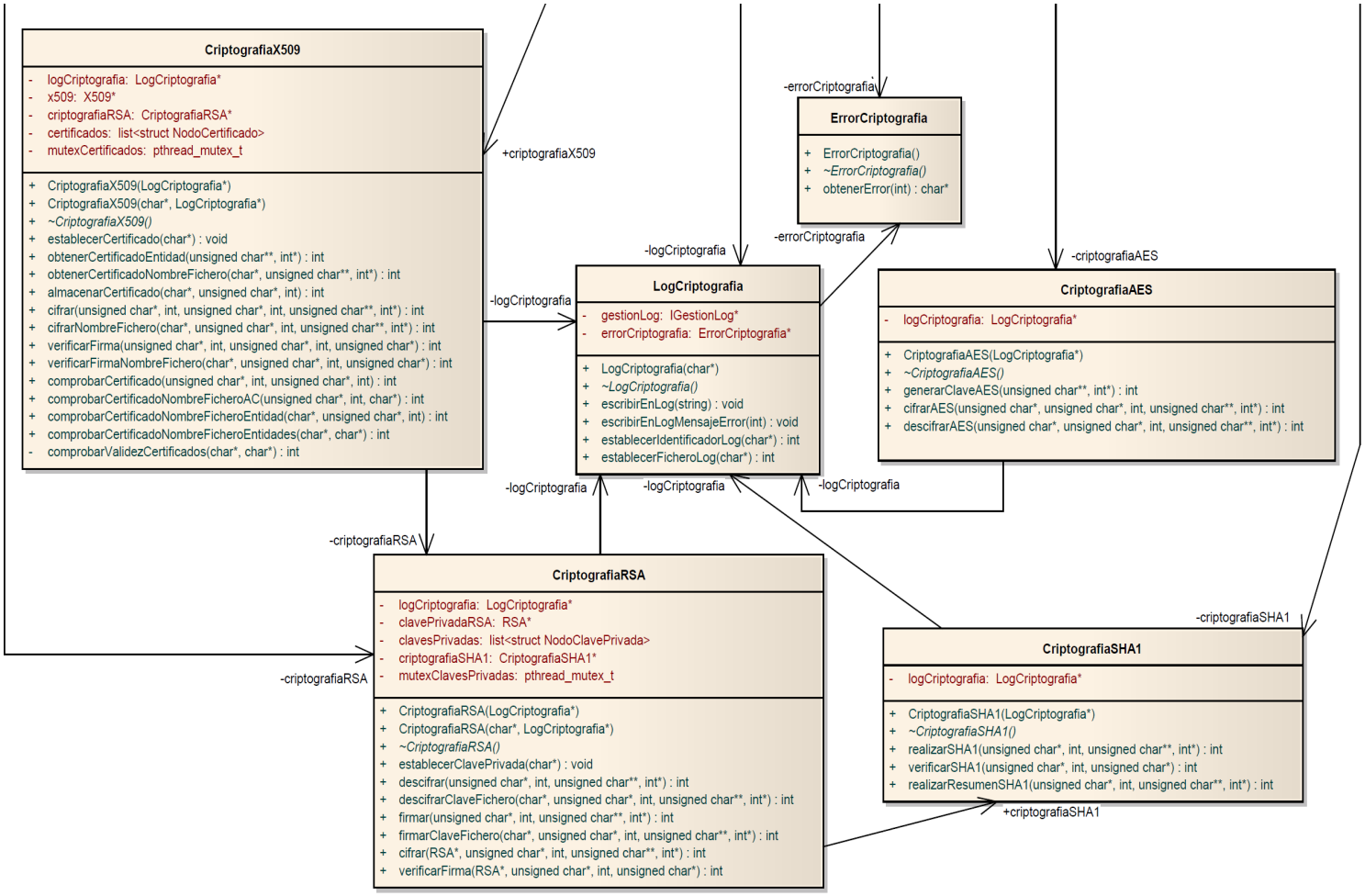

Ilustración 22: diagrama de clases 2 del componente Criptografia.

4.1.1.1.7. Diseño del componente Log.

En la Ilustración 23 se presenta el diseño del componente *Log*. Como se especificó en la sección de análisis, el componente *Log* proporciona las operaciones necesarias para la creación y gestión de logs.

Como se puede observar, la interfaz con los servicios ofrecidos por este componente es *IGestionLog*. También se muestra la clase *GestionLog*, que implementa la interfaz de este componente ofreciendo una fachada (patrón *Facade* [Bibliografía-3]) para el acceso a la funcionalidad ofrecida por las distintas clases de este componente.

La primera clase en la que se implementa la funcionalidad de este componente es *OperacionesLog*, la cual se encarga principalmente de crear las entradas de log que se almacenarán posteriomente en la clase *EscrituraLog*. Las operaciones que se pueden destacar de esta clase son:

- *establecerIdentificadorLog*: señalar el identificador que se utilizará para reconocer las entradas del log almacenadas por una determinada entidad (*identificadorLog*).
- *escribirEnLog*: crear una entrada del log, cuyo formato sea conforme a lo especificado en los requisitos del sistema, e invocar a la operación *almacenarMensaje* de la clase *escrituraLog*, para almacenarlo posteriormente en el log.

EscrituraLog es un módulo escrito en C, que proporciona operaciones encargadas de almacenar las entradas del log, creadas en la clase *OperacionesLog*, en el correspondiente fichero.

Con el objetivo de evitar que la inserción de entradas en el log implique perdida de rendimiento en la realización de operaciones criptográficas, en esta clase se crea un hilo de ejecución independiente que realice el almacenaje en el log de manera asíncrona. A continuación, se indican las principales operaciones que componen esta clase:

- *almacenarMensaje*: insertar la entrada del log a almacenar en la lista de entradas pendientes de escribir. De esta manera, la entrada queda almacenada en memoria, evitándose el retardo por la escritura en disco.
- *escribirEnLog*: operación ejecutada por el hilo de ejecución independiente, en el que periódicamente se escriban las entradas pendientes a almacenar en el log. Con esto se consigue que las entradas se vayan almacenando en disco, de manera asíncrona, sin necesidad de que el hilo de ejecución principal tenga que sufrir retardo en su realización.

Con el objetivo de incrementar la funcionalidad de este componente, se debe proporcionar la capacidad de que existan diversos hilos de ejecución distintos, cada uno de ellos encargado de almacenar entradas en un fichero de log determinado.

Finalmente, considerando que varios procesos distintos pueden escribir de manera concurrente en el log, se debe hacer uso de mecanismos que permitan el acceso controlado al mismo.

Por último, aunque en el sistema que se está desarrollando, este componente tiene como objetivo el almacenaje de las operaciones criptográficas realizadas, se puede reutilizar para la creación de log en cualquier otra aplicación.

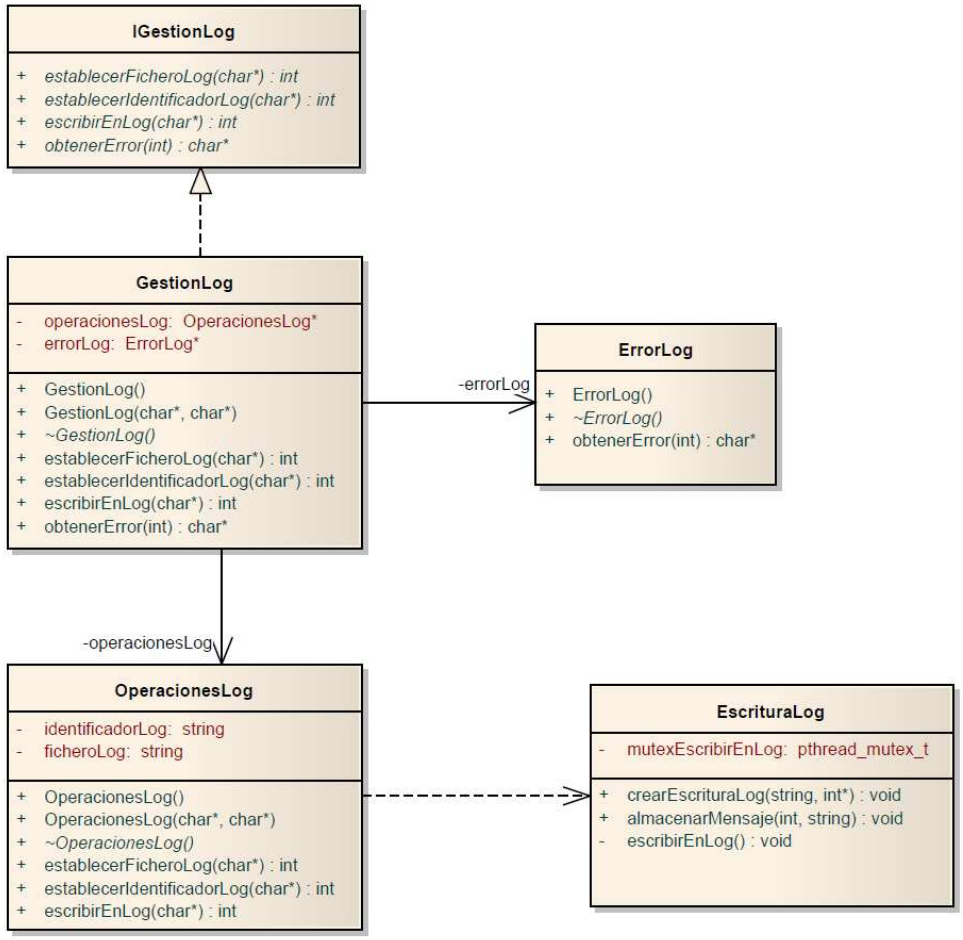

Ilustración 23: diagrama de clases del componente Log.

4.1.1.1.8. Diseño del componente InformacionSimulador.

En la Ilustración 24 se muestra el diagrama de clases del componente *InformacionSimulador*. Como se puede recordar, el propósito de este componente es el de obtener información proveniente del simulador, necesaria por las entidades para desempeñar su rol en el sistema.

Debido a la simplicidad de la funcionalidad de este componente, este únicamente está formado por la interfaz que ofrece a sus clientes, *IInformacionSimulador*, y la clase que la implementa, *InformacionSimulador*. Esta interfaz ofrece las operaciones *getID*, *getPosX* y *getPosY*, cuyo objetivo es obtener respectivamente el identificador de la entidad (*id*), y las coordenadas X (*posX*) e Y (*posY*) en la que se localiza. Además, es necesario indicar, que la información relativa a *posX* y *posY* debe ser obtenida a partir de los ficheros de configuración creados por el simulador en cada simulación, e *id* de la información proporciona por el kernel de NCTUns [Referencias-1].

Como se puede comprobar, este componente también puede ser reutilizado por nuevas entidades que pudiesen participar en una ampliación del protocolo, ya que todas las entidades comparten esta información que se obtiene del simulador.

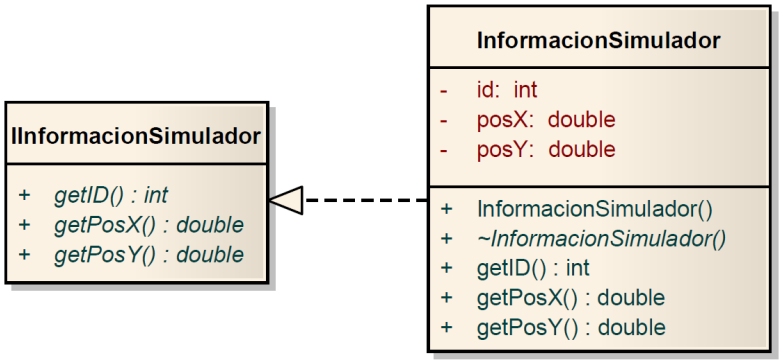

Ilustración 24: diagrama de clases del componente InformacionSimulador.

4.1.1.1.9. Diseño del componente ObtencionConfiguracion.

En la Ilustración 25 se presenta el diseño de clases del componente *ObtencionConfiguracion*. Tal y como se especificó en la sección de análisis, el componente *ObtencionConfiguracion* tiene como objetivo proporcionar soporte para la obtención de variables de configuración que se encuentren almacenadas en un dispositivo de almacenamiento concreto.

En primer lugar, se encuentra la interfaz proporcionada por este componente, *IObtencionConfiguracion*, y una clase que la implementa, *ObtencionConfiguracion*, que del mismo modo que se ha visto en componentes anteriores, proporciona una fachada (patrón *Facade* [Bibliografía-3]) encargada de coordinar el acceso a la distinta funcionalidad ofrecida por el componente.

En este componente también se sitúa la clase *LecturaConfiguracion*, cuyo objetivo es el de recuperar los parámetros de configuración almacenados en un dispositivo de almacenamiento concreto y de implementar operaciones para ofrecer dichos parámetros a los clientes de este componente. En esta clase se distingue principalmente la operación *leerConfiguración*, que tiene como propósito recuperar los parámetros de configuración del dispositivo de almacenamiento en el que se encuentran, delegando en la operación *leerConfiguracion* de *ITipoConfiguracion*, cuyo objetivo se especificará posteriormente. También se distinguen las operaciones *obtenerParametro*, *obtenerNumeroEntero*, *obtenerNumeroDecimal*, *obtenerPuerto* y *obtenerIP*, las cuales tienen como objetivo obtener de manera respectiva el valor de un parámetro de configuración cualquiera, de un número entero, de un número decimal, de un puerto y de una IP, de la lista *parametrosConfiguracion*, en la cual se deben haber insertado previamente los correspondientes parámetros obtenidos en la ejecución de la operación *leerConfiguracion*.

Como se puede comprobar, en la lista *parametrosConfiguracion* se deben almacenar estructuras de tipo *ParametroConfiguracion*, por lo que el programador debe crear la misma de manera que su contenido esté formado por el nombre y valor del parámetro de configuración.

Con el fin de poder realizar la recuperación de los parámetros de configuración de distintos tipos de dispositivos de almacenamiento, sin tener que realizar demasiados cambios en el componente, se hace uso del patrón *Strategy* [Bibliografía-3], cuya interfaz es *ITipoConfiguracion*, siendo ésta la

interfaz que debe ser implementada por las distintas estrategias encargadas de realizar la carga de un soporte de almacenamiento concreto.

Como se puede observar en la Ilustración 25, la única estrategia implementada es la obtención de los parámetros de un fichero (debido a que en el sistema a desarrollar las entidades deben obtener los parámetros de un fichero), pero puede ser la misma modificada, o incluir nuevas estrategias, sin afectar a la estructura y comportamiento del componente.

Otra de las clases de este componente es *ComprobacionConfiguracion*, cuyo objetivo es servir de soporte al componente, así como a clientes que quieran hacer uso de su funcionalidad, para verificar que el formato y valor de ciertos parámetros de configuración son concordes a un tipo concreto. A continuación, se especifican las operaciones de esta clase:

- *comprobarNumeroEntero*: determinar si el valor de la variable introducida como parámetro es de tipo entero.
- *comprobarNumeroDecimal*: verificar que el valor del parámetro especificado es de tipo decimal.
- *comprobarPuerto*: comprobar que el valor del parámetro es de tipo entero, y que debe encontrarse en el intervalo entre 0 y 65535.
- *comprobarIP*: comprobar que el valor del parámetro especificado tiene el formato "<*valor*>.<*valor*><*valor*>.<*valor*>", siendo *valor* un número entero entre 0 y 255.

Es necesario indicar que las operaciones *obtenerNumeroEntero*, *obtenerNumeroDecimal*, *obtenerPuerto* y *obtenerIP*, hacen uso respectivamente de las operaciones *comprobarNumeroEntero*, *comprobarNumeroDecimal*, *comprobarPuerto* y *comprobarIP* para verificar que el formato y valor del parámetro determinado pertenece al tipo correspondiente.

Por último, como se puede verificar, aunque este componente es utilizado por las entidades para obtener las variables de configuración que necesitan para su funcionamiento de un fichero, tiene la capacidad para ser reutilizado en cualquier otra aplicación que necesite de estos servicios.

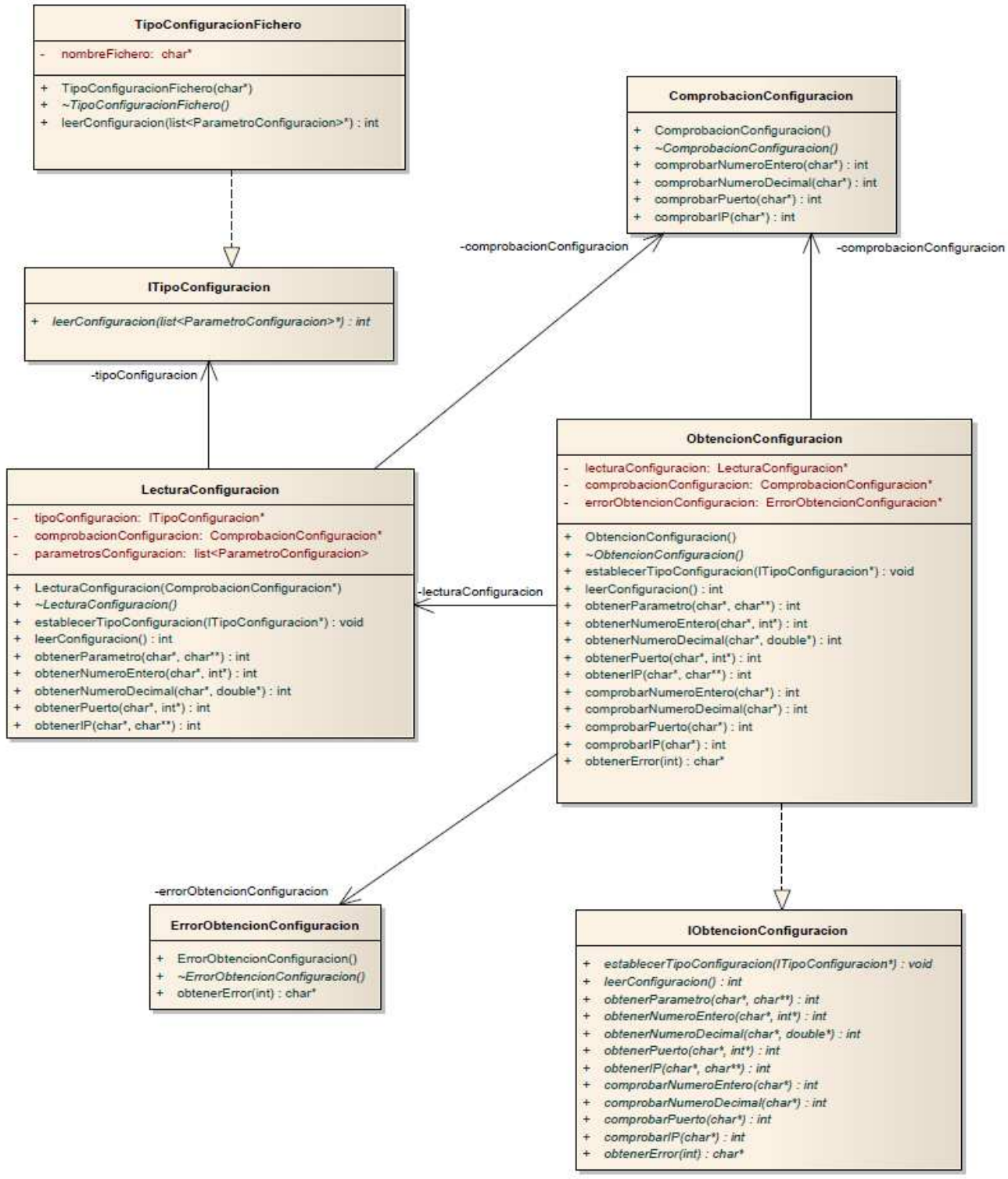

Ilustración 25: diagrama de clases del componente ObtencionConfiguracion.

4.1.1.1.10. Diseño del componente Comunicaciones.

En la Ilustración 26 se muestra el diagrama de clases del componente *Comunicaciones*. Como se puede recordar, el propósito de este componente es el abstraer distintos mecanismos de comunicación, con el fin de que los clientes que hagan uso de este componente puedan comunicarse con otras

entidades o aplicaciones, de la manera más transparente posible al mecanismo de comunicación utilizado.

Este componente ofrece dos interfaces distintas, *IRealizacionComunicaciones* e *IGestionComunicacion*. La interfaz *IRealizacionComunicaciones* proporciona los servicios que necesitan los clientes para establecer comunicación con otras entidades o escuchar en determinados puertos, e *IGestionComunicacion* se utiliza una vez se ha definido la comunicación, para realizar el envío y recepción de los mensajes correspondientes.

Como se comprueba en la Ilustración 26, la clase *RealizacionComunicaciones* implementa a *IRealizacionComunicaciones*, y del mismo modo que se ha hecho en otros componentes, proporciona una fachada de acceso (patrón *Facade* [Bibliografía-3]) a las distintas clases que implementan la funcionalidad del componente.

La clase *GestionComunicaciones* se debe encargar de establecer el mecanismo que se va a emplear en la comunicación, a través del uso de la operación *establecerComunicacion*, y de delegar la funcionalidad de comunicación en dicho mecanismo.

Resulta necesario destacar que, con el fin de fomentar la mantenibilidad y extensibilidad de este componente, se ha hecho uso del patrón *Strategy* [Bibliografía-3] para la utilización de los distintos mecanismos de comunicación. La interfaz de este patrón es *ITipoComunicaciones*, que define las siguientes operaciones:

- *crearComunicacion*: establecer la comunicación con una máquina con IP y puerto especificados como parámetro.
- *escucharPuerto*: dejar escuchando a la máquina local en un puerto determinado, a la espera de que se establezcan comunicaciones con ella.

Teniendo en cuenta que las comunicaciones que se emplean en el sistema son mediante sockets UDP, se debe crear la clase *ComunicacionesSocketUDP* para la implementación de ese mecanismo. Aún así, si en el futuro se modificase ese mecanismo, o se incorporasen nuevos que puedan implementar la interfaz *ITipoComunicaciones*, el resto del componente no se vería afectado por dichas actualizaciones.

Una vez se establezca una comunicación, ya sea de manera activa *(crearComunicacion*) o de manera pasiva (*escucharPuerto*), se le debe proporcionar al cliente de un objeto de tipo *IGestionComunicacion* para la realización de las operaciones propias de esa comunicación. Como se puede verificar, *IGestionComunicacion* es una interfaz que es implementada según el mecanismo de comunicación concreto utilizado. Por eso mismo, se debe crear la clase *ComunicacionSocketUDP* para la implementación del mecanismo de sockets UDP. A continuación, se describen las principales operaciones definidas en la interfaz *IGestionComunicacion*:

- *enviarMensaje*: enviar un mensaje a la entidad con la que se está comunicando.

- *recibirMensaje*: recibir un mensaje del nodo con el que se ha establecido la comunicación. Esta operación es bloqueante, de manera que el cliente que ejecuta la operación debe esperar de manera indefinida hasta la recepción del mismo.
- *recibirMensajeNoBloqueante*: realizar la misma funcionalidad que *recibirMensaje*, pero teniendo en cuenta que se debe esperar de manera definida un número determinado de segundos.
- *terminarComunicacion*: concluir la comunicación, liberando todos los recursos que hayan sido necesarios para su realización.

Finalmente, únicamente señalar el carácter reutilizable de este componente, el cual puede ser utilizado en cualquier aplicación para efectuar comunicaciones con otras, abstrayendo los mecanismos de comunicación concretos.

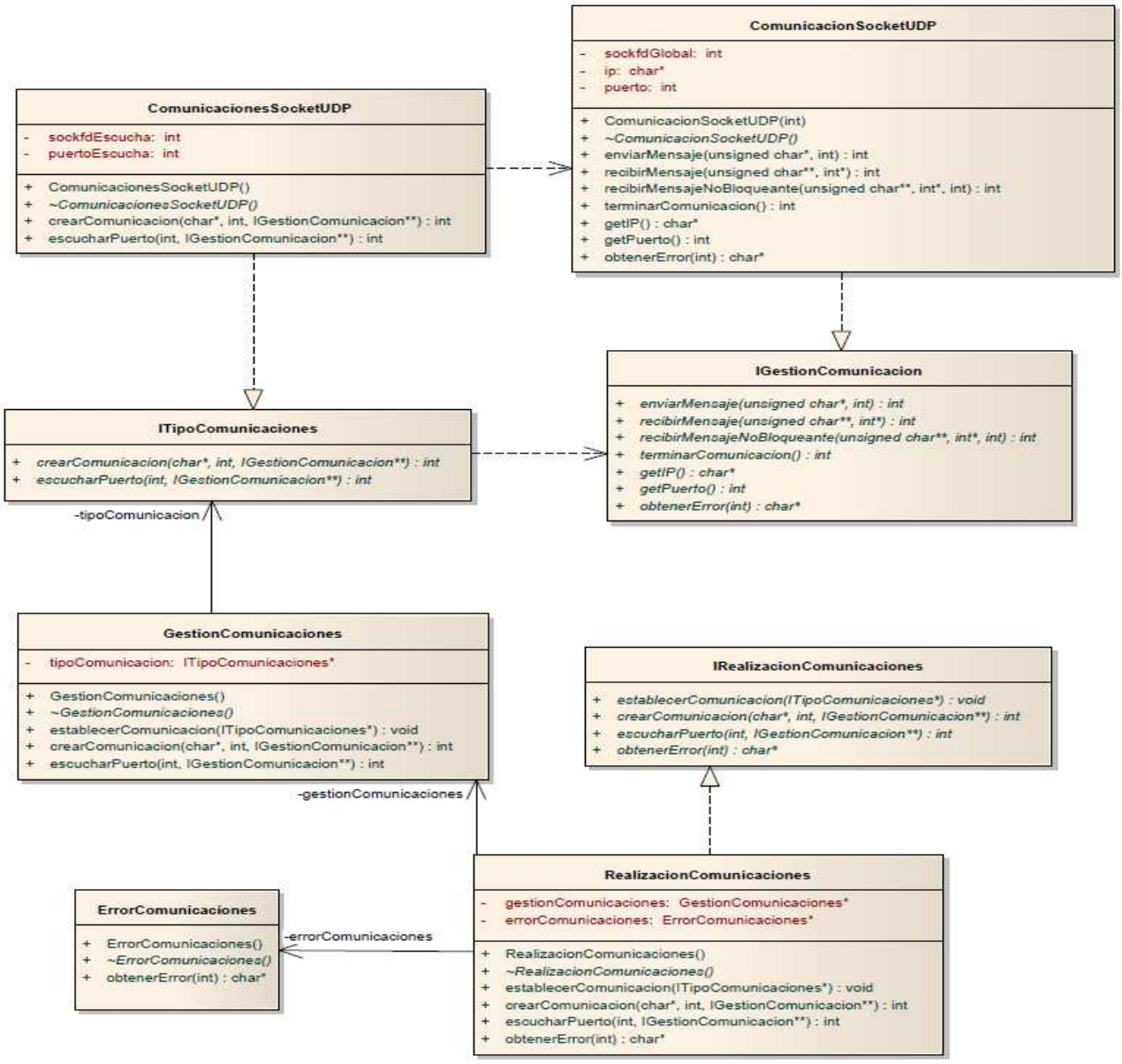

Ilustración 26: diagrama de clases del componente Comunicaciones.

4.1.1.1.11. Diseño del componente TomarMedidas.

El diseño de clases del componente *TomarMedidas* se presenta en la Ilustración 27. El propósito de este componente es el de proporcionar funcionalidad de soporte para la obtención de indicadores de rendimiento de protocolos, y el de enviar dichos indicadores al sistema de representación de los mismos.

La interfaz que proporciona este componente es *ITomarMedidas*, la cual es implementada por la clase *TomarMedidas*, que al igual que en componentes anteriores, proporciona una fachada (patrón *Facade* [Bibliografía-3]) para coordinar el acceso a la funcionalidad contenida en este componente distribuida en distintas clases.

La clase encargada de realizar las mediciones es *RealizarMedidas*. En esta clase se distingue la operación *medirBytesTransmitidos*, que tiene como propósito medir el número de bytes transmitidos en un cierto mensaje.

En esta clase se diferencian también las operaciones *comenzarMedirTiempoEnvio*, *comenzarMedirTiempoRespuesta* y *comenzarMedirTiempoComputacion*, las cuales se encargan de obtener respectivamente el instante de tiempo en el que se comienza a medir el tiempo de envío de un mensaje (almacenando el mismo en la lista *listaTiempoEnvio*), el instante en el que se empieza a medir el tiempo que se tarda en recibir la respuesta a un mensaje concreto (almacenándolo en la lista *listaTiempoRespuesta*) y el instante de tiempo en el que se empieza a medir el tiempo de computación de una operación criptográfica (almacenando el mismo en la lista *listaTiempoComputacion*). Respectivamente a estas operaciones, se encuentran *terminarMedirTiempoEnvio*, *terminarMedirTiempoRespuesta* y *terminarMedirTiempoComputacion*, cuyo propósito es obtener la diferencia de tiempo entre el instante en el que se ejecutan y la medida de tiempo obtenida en su respectiva operación anterior (*comenzarMedirTiempoEnvio*, *comenzarMedirTiempoRespuesta* y *comenzarMedirTiempoComputacion*). Además, también se distingue la operación *comenzarMedirTiempoRespuestaGeocast*, en la que se efectúa la misma funcionalidad que en *comenzarMedirTiempoRespuesta*, considerando que el mensaje enviado puede tener múltiples respuestas. Por tanto, si se hace uso de *comenzarMedirTiempoRespuestaGeocast* en lugar de *comenzarMedirTiempoRespuesta* para efectuar la medición, se pueden hacer múltiples mediciones de dicho tiempo de respuesta, en lugar de una sola.

Otras de las operaciones que se pueden diferenciar en la clase *RealizarMedidas* es *protocoloFinalizadoCorrectamente* y *protocoloFinalizadoErroneamente*, que se encargan de indicar que el protocolo ha finalizado, respectivamente, con éxito o fracaso; y las operaciones *aceptarParticipacionProtocolo* y *rechazarParticipacionProtocolo*, cuyo fin es señalar que una entidad, de manera respectiva, ha aceptado o rechazado su participación en el protocolo.

El motivo por el que los tiempos de envío, respuesta y computación se deben almacenar en listas es con el fin de incrementar la funcionalidad del componente, permitiendo la realización de varias mediciones simultáneas.

Como se puede comprobar, las listas *listaTiempoEnvio* y *listaTiempoComputacion* aceptan como elementos estructuras de tipo *NodoTiempo*, siendo tarea del programador crear la misma, de manera que en su contenido haya un identificador de la medición que está realizando y el tiempo en el que se comenzó a realizar la misma. Sin embargo, en la lista *listaTiempoRespuesta* se deben almacenar estructuras de tipo *NodoTiempoRespuesta*, cuyo contenido debe ser similar a *NodoTiempo* incluyendo el tipo de la medición de respuesta (si se ejecuta con *comenzarTiempoRespuesta* o *comenzarTiempoRespuestaGeocast*).

Otra de las clases de este componente es *EnviarMedidas*, la cual se encarga de crear los mensajes correspondientes que se enviarán al sistema de representación de indicadores. En esta clase se ubican las operaciones *enviarBytesTransmitidos*, *enviarTiempoEnvio*, *enviarTiempoRespuesta*, *enviarTiempoComputacion*, *enviarProtocoloFinalizado* y *enviarAceptacionProtocolo*, que tienen como objetivo crear el mensaje con el indicador respectivo del número de bytes transmitidos, tiempo de transmisión, tiempo de respuesta, tiempo de computación, éxito o fracaso en la finalización del protocolo, o aceptación o rechazo en la participación en el mismo.

Otra de las operaciones que componen la clase *EnviarMedidas* es *leerFicheroConfiguracion*, la cual se debe encargar de obtener de un fichero los parámetros IP y puerto que se necesitan para la comunicación con el sistema de representación de indicadores. Además, en caso de no especificarse dicha configuración, se deben proporcionar valores por defecto a dichos parámetros.

Teniendo en cuenta que el sistema de representación de indicadores debe ser desarrollado en Java, y cumpliendo con lo especificado en los requisitos de que el formato de estos mensajes debe ser NVT ASCII, se expone el formato que deben seguir los mismos.

ID TIEMPO INDICADOR\r\n\r\n

- *ID*: identificador del indicador obtenido. Este valor toma el valor 1 para el indicador de los bytes transmitidos, 2 para el tiempo de envío, 3 para el tiempo de respuesta, 4 para el tiempo de computación, 5 para indicar el éxito o fracaso en la finalización del protocolo y 6 para la aceptación o rechazo en su participación.
- *TIEMPO*: instante de tiempo en el que se obtiene el indicador. Este tiempo debe ser 0 al crearse el componente.
- *INDICADOR*: valor del indicador obtenido. Este valor puede corresponder con el número de bytes transmitidos, el número de milisegundos empleados en el envío, el número de milisegundos empleados en la respuesta, el número de milisegundos dedicados a la computación de las operaciones criptográficas, valor 1 o 0 si el protocolo ha finalizado de forma correcta o incorrecta, y valor 1 o 0 si se ha aceptado o rechazado la participación en el protocolo.

En lo que respecta a *EnvioMensajesAnalizador*, se trata de un módulo desarrollado en C, en el que se debe crear un hilo de ejecución independiente encargado de la transmisión, de manera periódica, de los mensajes con los indicadores al sistema de representación de los mismos. De esta manera,

cuando se crea un mensaje a enviar al sistema de representación, se debe almacenar el mismo en una lista de mensajes pendientes, de forma que el hilo independiente asíncronamente al hilo de ejecución principal realice la transmisión de los mismos. Con esto se consigue evitar pérdida de rendimiento en la ejecución de las entidades por la obtención de indicadores. A continuación, se indican las principales operaciones que componente este módulo:

- *almacenarMensaje*: almacenar un determinado mensaje en la lista de mensajes pendientes, los cuales deben ser enviados posteriormente por el hilo.
- *enviarMensajesAnalizador*: operación ejecutada por el hilo encargado del envío de los mensajes al sistema de representación de indicadores.

Finalmente, es necesario añadir que este componente puede ser reutilizado para obtener indicadores de rendimiento de cualquier protocolo de envío de mensajes, no limitándose al que va a ser desarrollado en el sistema.

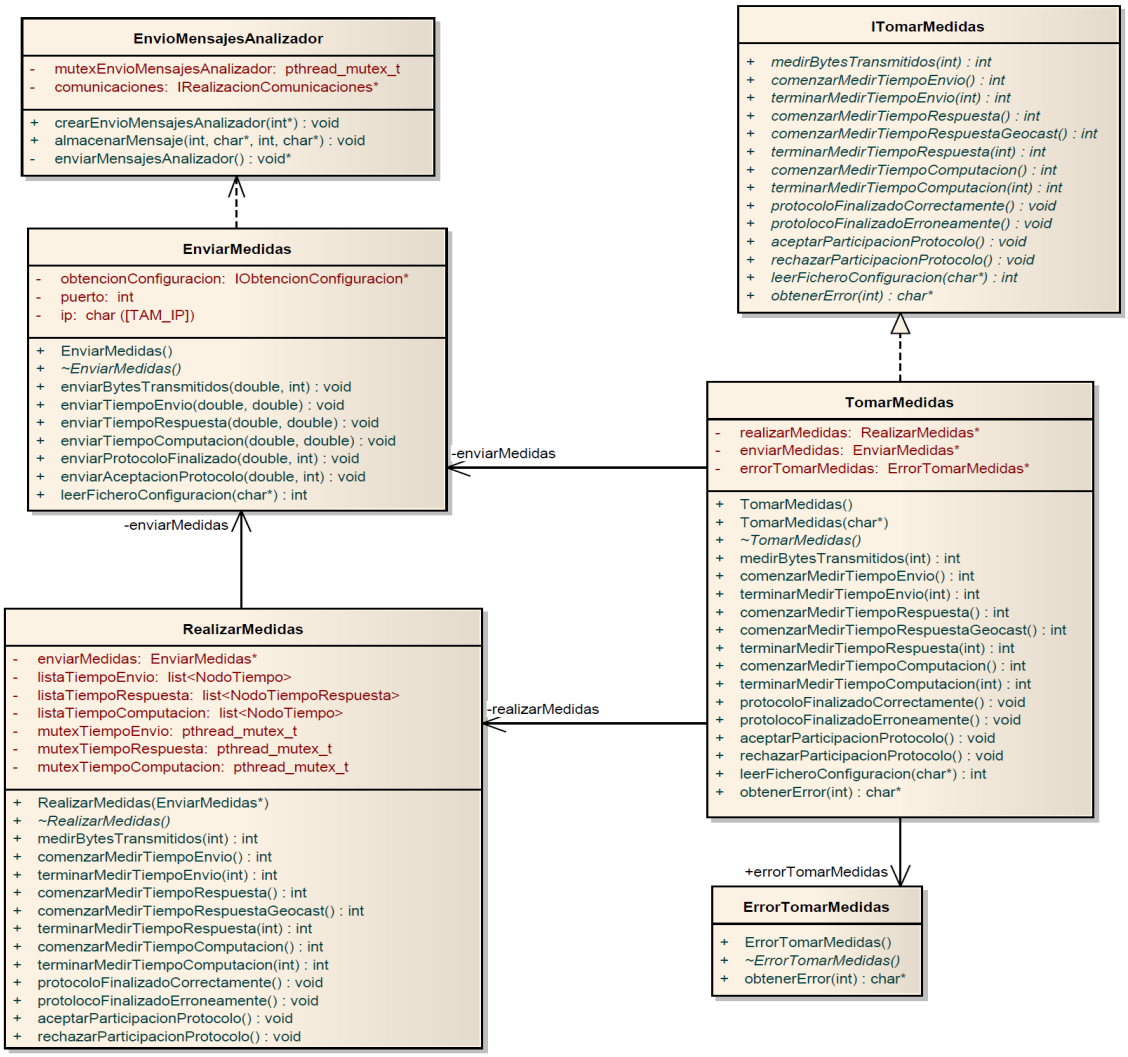

Ilustración 27: diagrama de clases del componente TomarMedidas.

4.1.1.1.12. Diseño del componente ComunicacionInterfaz.

En la Ilustración 28 se muestra el diagrama de clases del componente *ComunicacionInterfaz*. Como se puede recordar de la sección de análisis, el objetivo de este componente es el de ofrecer soporte a una RSU para enviar información asociada al intercambio de los mensajes del protocolo con los vehículos, así como el contenido de su *beacon*, al sistema de visualización de mensajes.

Teniendo en cuenta que el desarrollo de este componente es realizado por el Proyecto Fin de Carrera [Bibliografía-2] destinado, entre uno de sus objetivos, a la creación del sistema de visualización de mensajes, únicamente se van a especificar las operaciones definidas en la interfaz *IEnlaceComponente* ofrecida por este componente para hacer uso de sus servicios. Cada una de estas operaciones se encarga de enviar información determinada al sistema de visualización de mensajes, siendo para cada operación dicha información la siguiente:

- *recibidaEvidencia*: información asociada a la recepción en una RSU por parte del vehículo infractor, del mensaje con una evidencia de respuesta a una notificación de sanción.
- *procesarInfoEstadoRSU*: información contenida en el *beacon* enviado por una RSU.
- *enviadaNotificacion*: información relacionada con el envío de un mensaje con una notificación de sanción, de una RSU al vehículo infractor.
- *recibidoMensajeNE_RSU*: información asociada a la recepción en una RSU de un mensaje con un testimonio aportado por un testigo, procedente de un vehículo testigo o no equipado.
- *enviadoMensajeDesdeNE_RSU*: información relacionada con el envío por parte de una RSU a un vehículo de tipo no equipado o infractor, de un mensaje con un testimonio creado por un testigo.

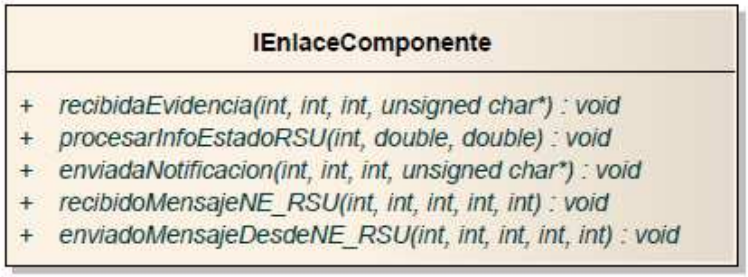

Ilustración 28: diagrama de clases del componente EnlaceComponente.

4.1.1.2. Diseño detallado de clases del sistema de representación de indicadores.

Antes de proceder al diseño detallado de clases de cada uno de los componentes del sistema de representación de indicadores, al igual que se ha realizado al comienzo del apartado 4.1.1.1, en la Ilustración 29, se presenta nuevamente la arquitectura de esta parte del sistema, pero indicando el número de subapartado en el que se describe cada componente:

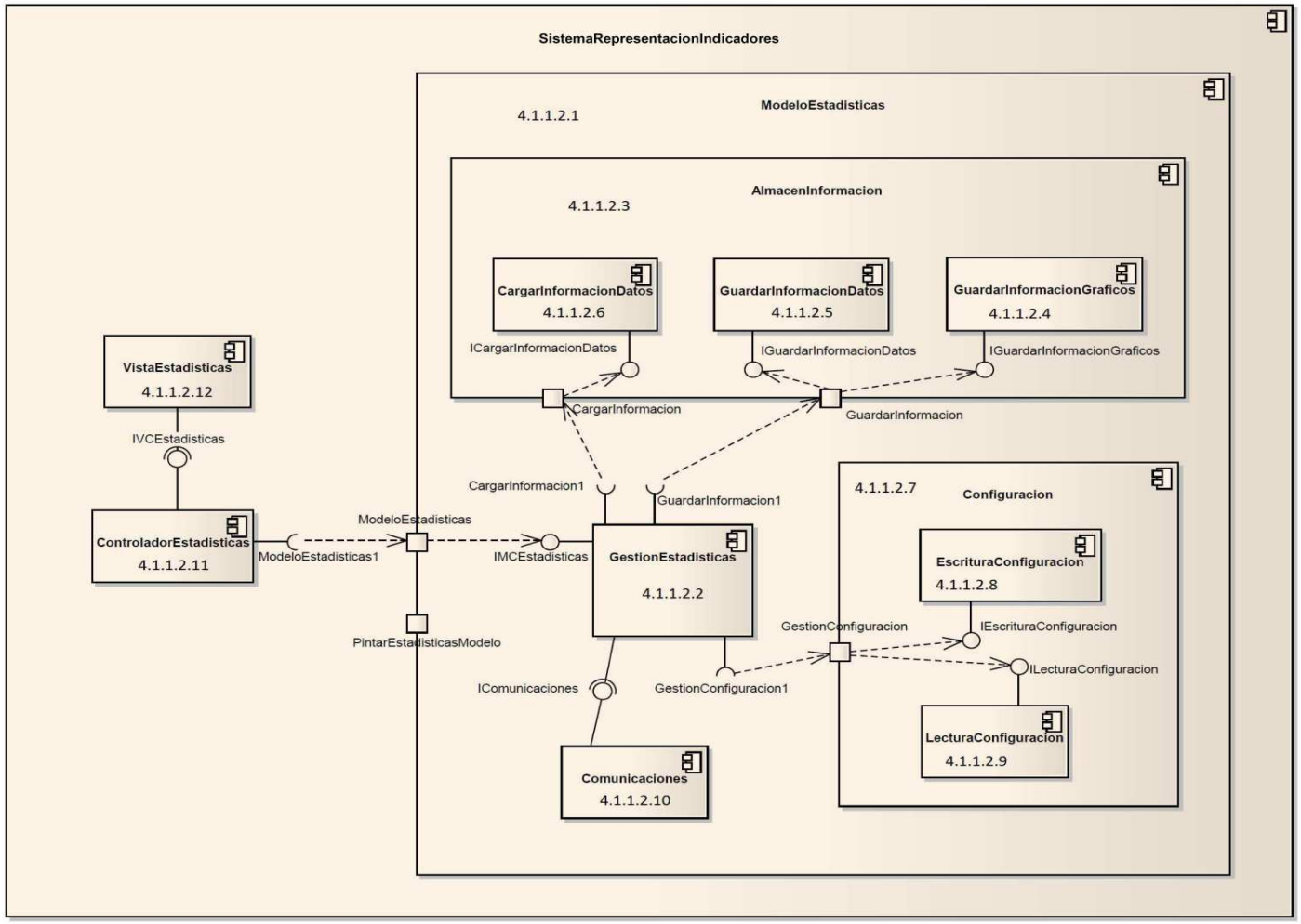

Ilustración 29: esquema especificación de componentes del sistema de representación de medidas.

4.1.1.2.1. Diseño del componente ModeloEstadisticas.

En la Ilustración 30 se presenta el diagrama de clases del componente *ModeloEstadisticas*. Como se especificó en la descripción de la arquitectura del sistema, este componente se corresponde con el Modelo del estilo MVC [Bibliografía-3], en el cual se engloban todos los componentes que componen la lógica de negocio del sistema de representación de indicadores.

En este diagrama se observan las clases *ModeloEstadisticas* y *PintarEstadisticasModelo*, las cuales implementan los puertos del mismo nombre incluidos en el componente.

El puerto *ModeloEstadisticas* proporciona una abstracción en la utilización de los servicios que ofrecen los componentes internos al componente *ModeloEstadisticas*, de manera que se pueda hacer uso de los mismos sin necesidad de conocer la estructura interna del componente.

El puerto *PintarEstadisticasModelo* tiene como propósito servir de intermediario en la utilización de los servicios proporcionados en la interfaz *IPintarEstadisticas* del componente *VistaEstadisticas*, por el componente *GestionEstadisticas*. El motivo por el que se realiza esto es para evitar que el componente interno *GestionEstadisticas* pueda acceder a otro componente que se encuentra fuera del ámbito de su componente superior *ModeloEstadisticas*.

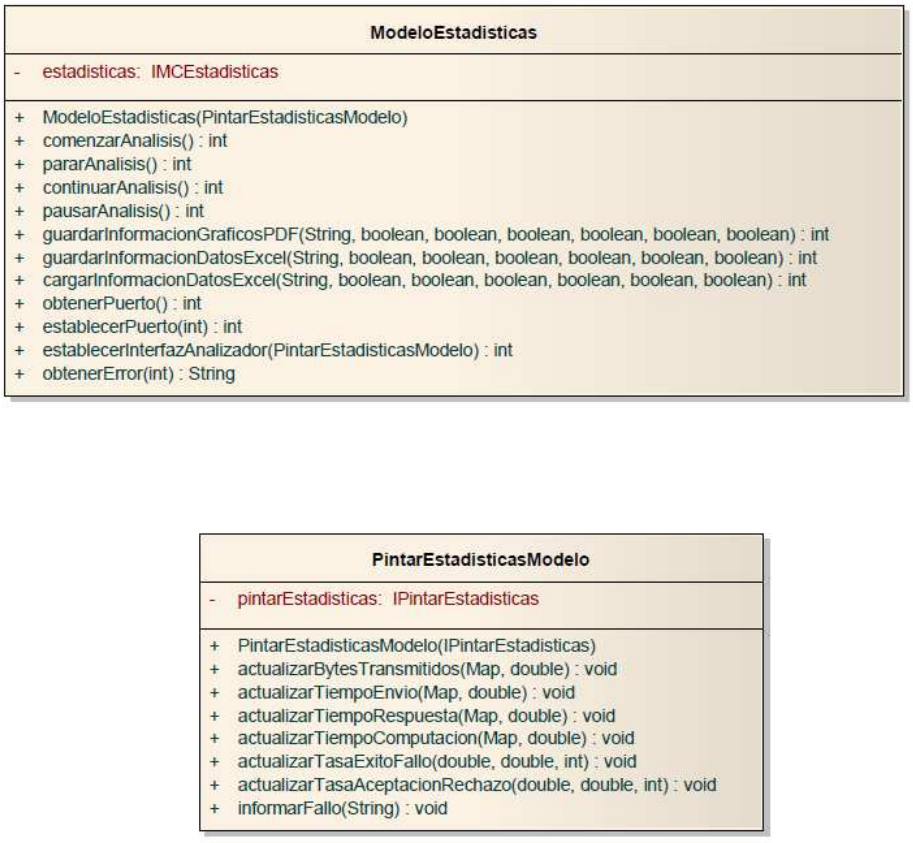

Ilustración 30: diagrama de clases del componente ModeloEstadisticas.

4.1.1.2.2. Diseño del componente GestionEstadisticas.

El diseño de clases del componente *GestionEstadisticas* se presenta en la Ilustración 31. Como se puede recordar de la sección de análisis, el objetivo de este componente es el de recibir los indicadores enviados al sistema actualizándolos adecuadamente, de realizar las operaciones de gestión del sistema, y de efectuar las operaciones de almacenaje y carga de la información en PDF o Excel 2003.

En la Ilustración 31, se puede contemplar la interfaz ofrecida por este componente, *IMCEstadisticas*, y la clase que la implementa, *GestionEstadisticas*, la cual actúa como fachada (patrón *Facade* [Bibliografía-3]) para coordinar el acceso a la funcionalidad distribuida entre las distintas clases del componente.

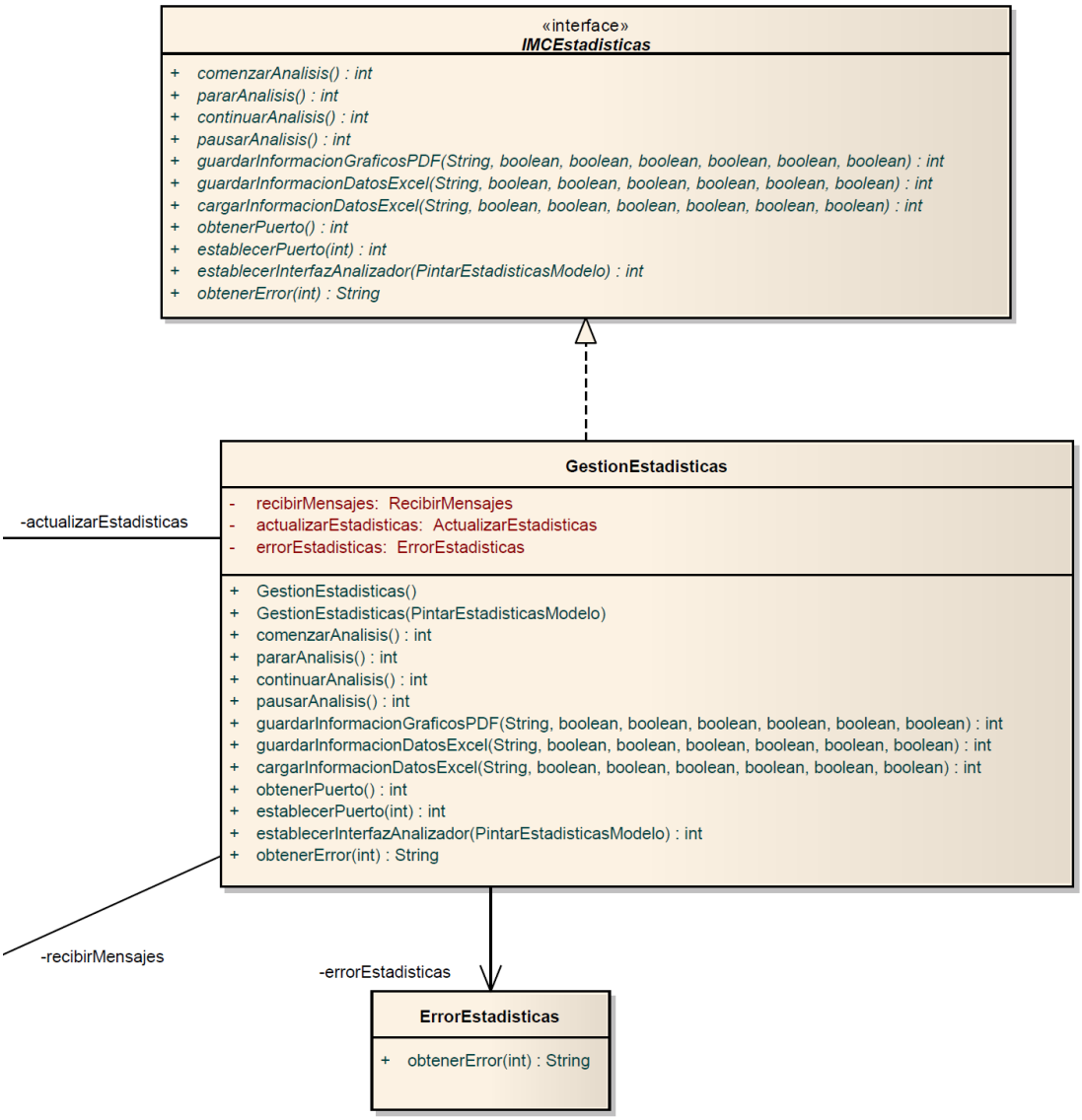

Ilustración 31: diagrama de clases 1 del componente GestionEstadisticas.

Como se puede observar en la Ilustración 32, se presenta la clase *RecibirMensajes*, cuyo propósito consiste en recibir los mensajes enviados al

sistema de representación de indicadores y de ejecutar las operaciones correspondientes en función de dichos envíos. Para ello, se debe crear un hilo de ejecución independiente encargado de la recepción de los mensajes, de manera que cada vez que se reciba uno, se actualicen los indicadores correspondientes con su contenido, haciendo uso del patrón *Observer* [Bibliografía-3]. Otro de los objetivos de esta clase es la implementación de las operaciones para la gestión del hilo receptor de los mensajes. En esta clase se distinguen las siguientes operaciones:

- *comenzarAnalisis*: arrancar el hilo encargado de la recepción de los mensajes. Previamente a su arranque, se debe reiniciar el sistema eliminando los posibles valores de la información de rendimiento obtenida de ejecuciones anteriores.
- *pararAnalisis*: parar el hilo receptor de los mensajes con los indicadores. Ejecutada esta operación, la única opción disponible que se debe permitir para arrancar nuevamente el hilo es *comenzarAnalisis*.
- *continuarAnalisis*: arrancar el hilo destinado a recibir los mensajes. Esta operación únicamente se puede realizar si se ha ejecutado la operación *pausarAnalisis*.
- *pausarAnalisis*: parar el hilo encargado de la recepción de los mensajes. Ejecutada esta operación, se puede arrancar nuevamente el hilo sin necesidad de eliminar indicadores obtenidos en ejecuciones previas (*continuarAnalisis*).
- *obtenerPuerto*: obtener el valor del puerto por el que se van a recibir los mensajes con los indicadores, del fichero de configuración correspondiente. En caso de que no se pueda recuperar dicho puerto del fichero, se debe proporcionar un valor por defecto al mismo.

- *establecerPuerto*: actualizar el valor del puerto de escucha de los mensajes. Además, se debe actualizar el fichero de configuración con el valor del nuevo puerto.

Otra de las clases que componen este componente es *ActualizarEstadisticas*, cuyo objetivo debe ser el de actualizar los valores correspondientes de los indicadores de rendimiento, y de transmitirlos a la interfaz gráfica para su representación. Para ello hace se hace uso de la operación *actualizarEstadisticas*, que tiene como objetivo la actualización del valor de los indicadores de rendimiento correspondientes con el tipo del indicador recibido en un determinado mensaje, considerando el valor del mismo, y de la operación *run*, la cual es ejecutada por un hilo independiente creado para representar, periódicamente, la información de los indicadores de rendimiento almacenados en esta clase, en la interfaz gráfica del sistema. El motivo por el que se crea este hilo es para controlar la actualización de la interfaz del sistema, de manera que se representen los datos a una velocidad adecuada, pero sin saturar a la misma.

Otro de los objetivos de la clase es la implementación de la operación *guardarInformacionGraficosPDF*, cuyo objetivo debe ser el almacenaje en PDF de los gráficos representados en la interfaz del sistema, así como la media de los indicadores del número de bytes transmitidos, tiempo de envío, tiempo

de respuesta y tiempo de computación; y las tasas de éxito y fracaso en la finalización del protocolo, y las tasas de aceptación o rechazo en la participación en el mismo. También debe incluirse en el fichero PDF, el número de medidas recibidas de cada tipo.

Las últimas operaciones que caben destacar de la clase *ActualizarEstadisticas* son *guardarInformacionDatosExcel* y *cargarInformacionDatosExcel*, las cuales se deben encargar, de manera respectiva, del almacenaje o carga de la información de los indicadores en un fichero Excel 2003.

La última clase que falta por describir de este componente es *DatosEstadisticas*, la cual tiene como propósito procesar los mensajes recibidos en el sistema comprobando que su formato y contenido son correctos (de acuerdo a lo especificado en el apartado 4.1.1.1.11). Además, si se comprueba que un determinado mensaje es correcto, se debe asignar el valor correspondiente al instante de tiempo en el que fue obtenido el indicador (*tiempo*) y el valor del indicador en función del tipo del mismo.

Como se puede comprobar, en este componente se encuentra la lógica que define el funcionamiento del sistema de representación de indicadores, sin tener en cuenta detalles de interfaz, pudiendo por tanto modificar el diseño de la misma sin afectar a este componente.

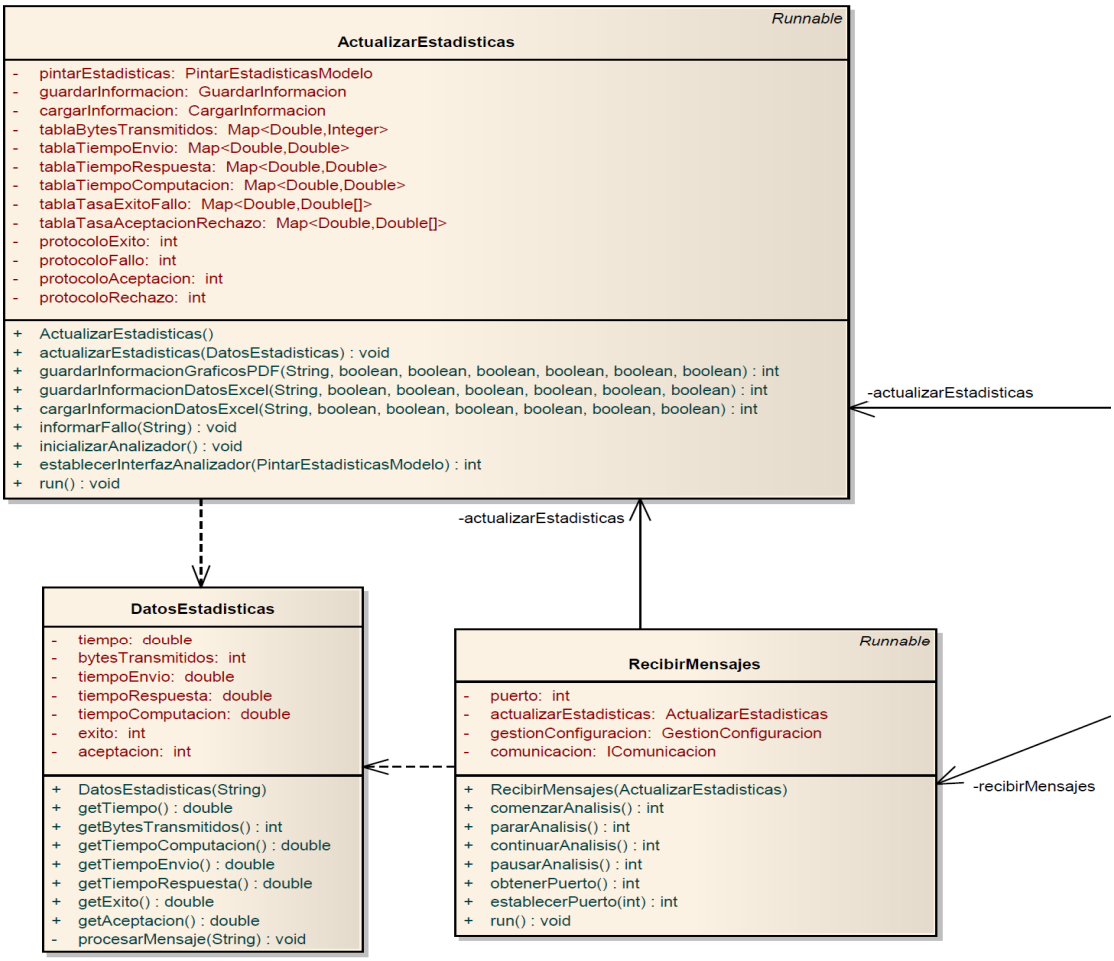

Ilustración 32: diagrama de clases 2 del componente GestionEstadisticas.

4.1.1.2.3. Diseño del componente AlmacenInformacion.

El objetivo del componente *AlmacenInformacion* es encapsular la funcionalidad ofrecida por el componente *GuadarInformacionGraficos* para guardar gráficos y texto en soportes electrónicos diferentes, y la funcionalidad proporcionada por los componentes *GuardarInformacionDatos* y *CargarInformacionDatos* para almacenar y cargar, respectivamente, datos en formatos distintos de hojas de cálculo.

Tal y como se aprecia en la Ilustración 33, las clases que componen este componente son *CargarInformacion* y *GuardarInformcion*, las cuales se corresponden con los puertos que pertenecen al mismo.

El puerto *CargarInformacion* se utiliza para abstraer el uso de los servicios ofrecidos por el componente *CargarInformacionDatos*, de manera que el cliente de los mismos no tenga necesidad de conocer la estructura interna del componente.

En lo que respecta al puerto *GuardarInformacion*, su objetivo es englobar los servicios proporcionados en las interfaces *IGuardarInformacionGraficos* e *IGuardarInformacionDatos*, de manera que se puedan utilizar los mismos sin necesidad de conocer los componentes concretos que los implementan.

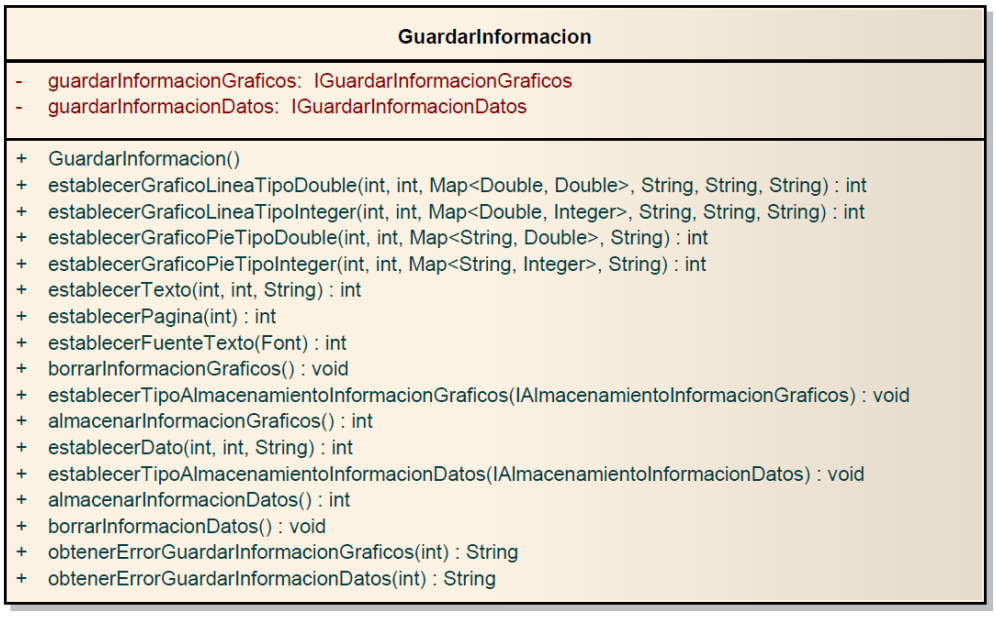

Ilustración 33: diagrama de clases del componente AlmacenInformacion.

4.1.1.2.4. Diseño del componente GuardarInformacionGraficos.

En la Ilustración 34 se presenta el diagrama de clases del componente *GuardarInformacionGraficos*. El objetivo de este componente es ofrecer soporte para el almacenamiento de gráficos relacionados con unos datos concretos, así como información textual, en distintos soportes electrónicos como PDF.

La interfaz ofrecida por este componente es *IGuardarInformacionGraficos*, la cual es implementada por la clase *GuardarInformacionGraficos*, que de la misma manera que en otros componentes descritos anteriormente, proporciona una fachada (patrón *Facade* [Bibliografía-3]) encargada de delegar las operaciones ejecutadas por los clientes del componente, en las distintas clases en las que se desarrolla su funcionalidad.

Otra de las clases de este componente es *OperacionesGuardarInformacionGraficos*, que debe tener como propósito el almacenaje, de manera previa, de los gráficos y textos que serán posteriormente almacenados en el soporte electrónico concreto. Las operaciones *establecerGraficoLineaTipoDouble* y *establecerGraficoLineaTipoInteger* se deben encargar de almacenar en la lista *listaGraficos*, de manera respectiva, un gráfico de tipo lineal con datos de tipo *Double* e *Integer*; y las operaciones *establecerGraficoPieTipoDouble* y *establecerGraficoPieTipoInteger* deben almacenar en la misma lista, un gráfico de tipo circular con datos de los mismos tipos. En lo que respecta al almacenamiento de texto, se debe hacer uso de la operación *establecerTexto*, para almacenar el mismo en la lista *listaTextos*.

En esta clase también se encuentra la operación *almacenarInformacionGraficos*, la cual se debe encargar de invocar a la operación *guardarInformacionGraficos* de la interfaz *IAlmacenamientoInformacionGraficos* con el contenido de las listas *listaGraficos* y *listaTextos*, con el objetivo de almacenar los gráficos y textos en el soporte electrónico especificado.

Para representar los gráficos y textos a almacenar, se hace uso, respectivamente, de las clases *Grafico* y *Texto*, las cuales, tal y como se puede apreciar en la Ilustración 34, heredan de la clase *ObjetoAlmacenar*.

En la clase *ObjetoAlmacenar* se especifican los atributos comunes de los gráficos y textos, como son la página en la que se deben almacenar, así como la localización dentro de la misma. Por otro lado, en la clase *Texto* se indica la fuente utilizada para su escritura, y en la clase *Grafico*, atributos propios de la misma como los datos que deben ser representados, el título del gráfico, los rótulos de los ejes X e Y, etc.

Además, con el objetivo de incrementar la mantenibilidad de este componente, se ha hecho uso del patrón *Strategy* [Bibliografía-3] para encapsular la funcionalidad destinada al almacenaje de la información en distintos soportes electrónicos. La interfaz de este patrón es *IAlmacenamientoInformacionGraficos* y la clase que la implementa es *AlmacenamientoInformacionGraficosPDF*, en la que se debe implementar una estrategia para el almacenaje de la información en formato PDF. No se han implementado más estrategias, debido a que en el sistema a desarrollar la

información de los gráficos se debe almacenar únicamente en formato PDF, pero éstas podrían incluirse sin provocar cambios en el resto del componente.

Antes de finalizar, resulta necesario destacar que este componente puede ser reutilizado en otras aplicaciones para realizar el almacenaje de gráficos que representen unos datos cualquiera, y el texto que se desee, en un soporte electrónico concreto.

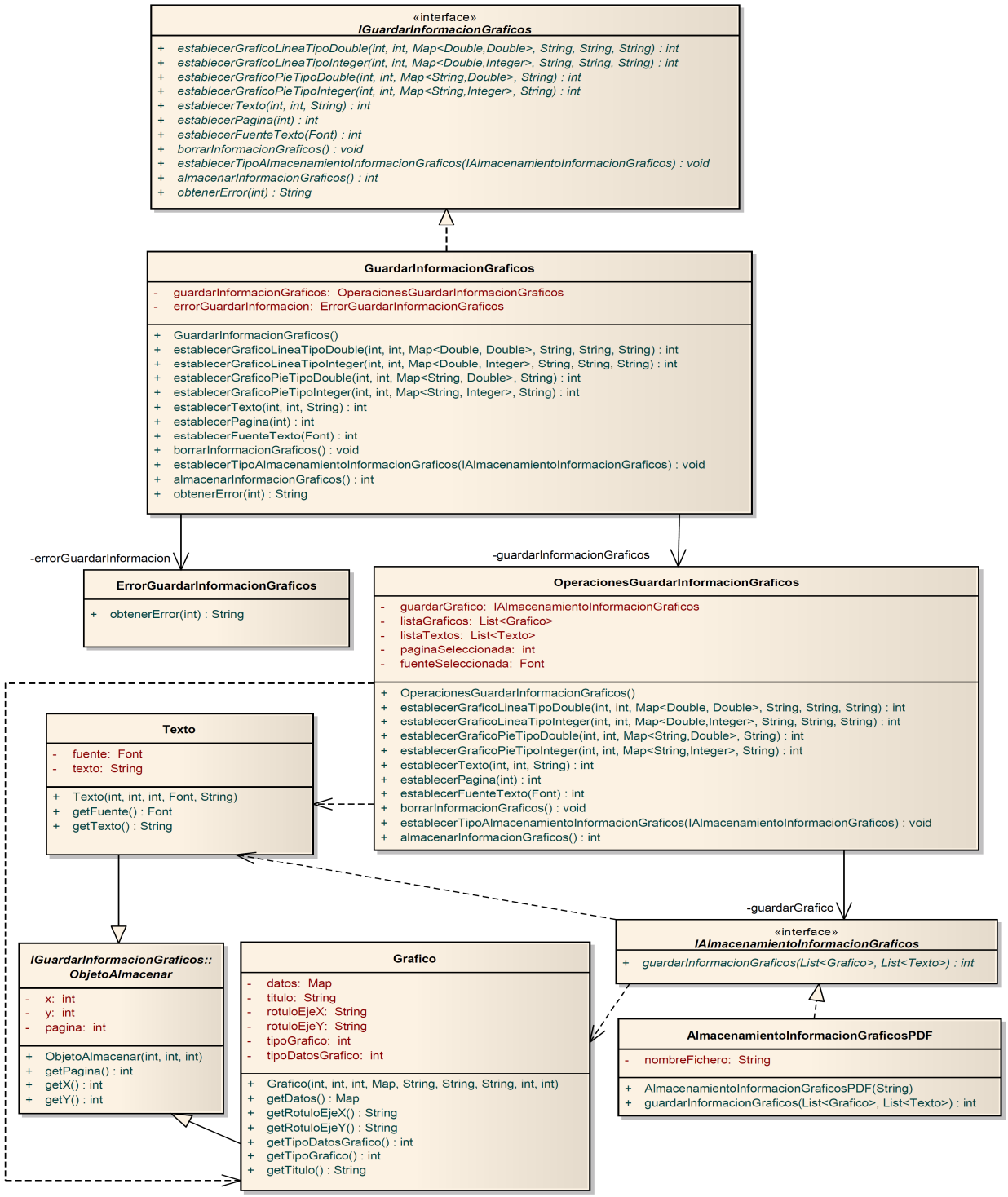

Ilustración 34: diagrama de clases del componente GuardarInformacionGraficos.

4.1.1.2.5. Diseño del componente GuardarInformacionDatos.

El diseño de clases del componente *GuardarInformacionDatos* se muestra en la Ilustración 35. Tal y como se especificó en el análisis, el componente *GuardarInformacionDatos* debe ofrecer soporte para el almacenaje de datos en formatos distintos de hojas de cálculo.

IGuardarInformacionDatos es la interfaz ofrecida por este componente, y *GuardarInformacionDatos* es la clase que la implementa, la cual actúa de fachada (patrón *Facade* [Bibliografía-3]) para coordinar el acceso al resto de clases del componente.

La clase *OperacioneGuardarInformacionDatos* tiene como misión la gestión de los datos que serán posteriormente almacenados en una hoja de cálculo. Para almacenar previamente los datos en memoria, antes de escribirlos en la hoja de cálculo, se debe hacer uso de la operación *establecerDato*, la cual se encarga de almacenar un dato concreto en la lista *datos*. Una vez se tienen todos los datos a almacenar, se debe utilizar la operación *almacenarInformacionDatos*, la cual se debe encargar de invocar a la operación *guardarInformacionDatos* de la interfaz *IAlmacenamientoInformacionDatos*, con el fin de almacenar los datos contenidos en la lista *datos* en el tipo de hoja de cálculo establecida.

Es necesario indicar que para representar el dato a almacenar en la hoja de cálculo se hace uso de la clase *Dato*, la cual contiene el dato concreto y las coordenadas de la celda en la que se debe ubicar.

Además, se ha hecho uso del patrón *Strategy* [Bibliografía-3] para elegir entre los distintos formatos de hoja de cálculo a almacenar. La interfaz de este patrón *es IAlmacenamientoInformacionDatos*, que contiene la operación *guardarInformacionDatos*, siendo ésta la operación en la que las clases que la implementen deben desarrollar la estrategia. Teniendo en cuenta que en el sistema a desarrollar únicamente hay que almacenar los datos de los indicadores en formato Excel 2003, la clase *AlmacenamientoInformacionDatosExcel* es la única que se tiene que implementar. No obstante, si en el futuro se añadiesen nuevos formatos de hoja de cálculo, el resto del componente no se vería afectado por ello.

Como se puede comprobar, este componente puede ser reutilizado en cualquier aplicación que requiera la necesidad de almacenar unos datos concretos en distintos formatos de hoja de cálculo.

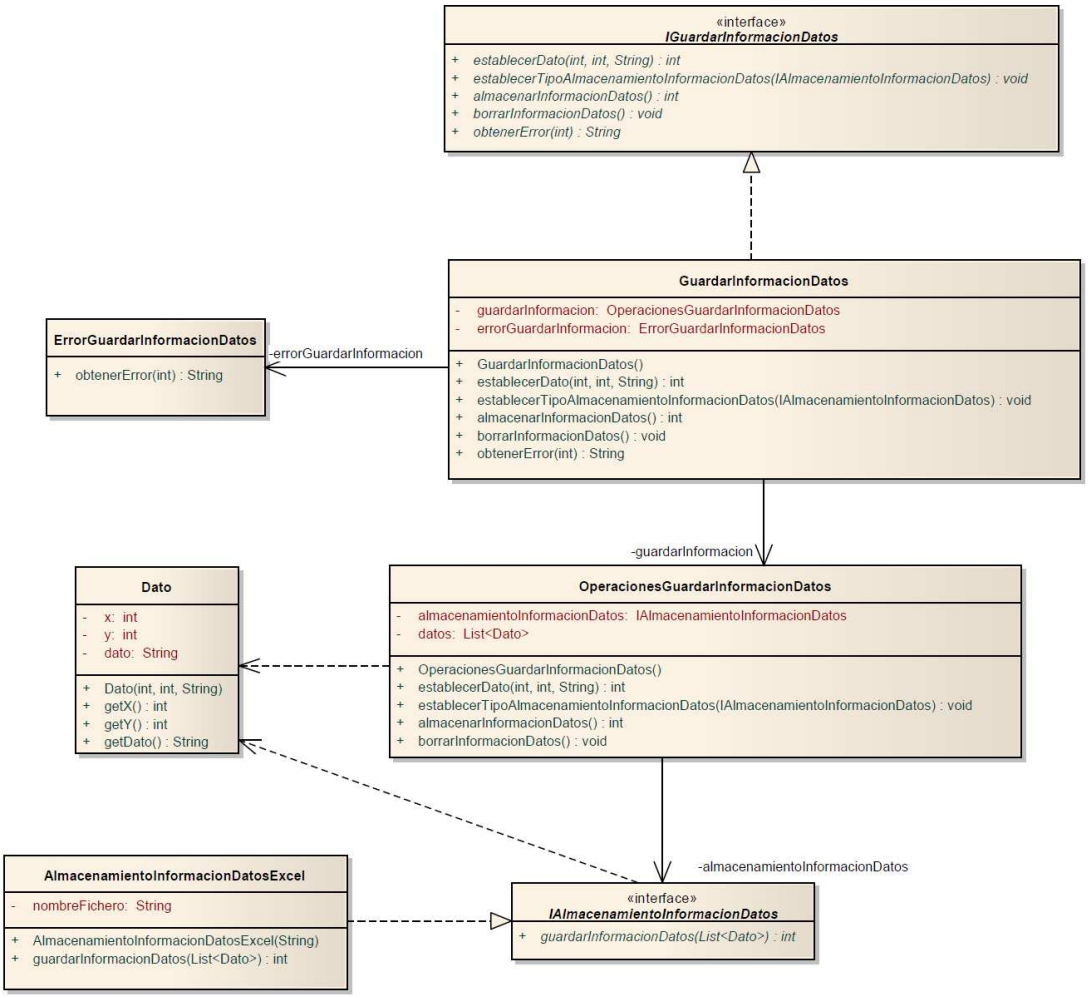

Ilustración 35: diagrama de clases del componente GuardarInformacionDatos.

4.1.1.2.6. Diseño del componente CargarInformacionDatos.

El diseño de clases del componente *CargarInformacionDatos* se presenta en la Ilustración 36. El objetivo de este componente es proporcionar funcionalidad a los clientes del mismo para obtener datos almacenados en formatos distintos de hojas de cálculo.

La interfaz que proporciona este componente es *ICargarInformacionDatos*, y es implementada por la clase *CargarInformacionDatos*, la cual actúa de fachada (patrón *Facade* [Bibliografía-3]) con el fin de controlar el acceso a la funcionalidad del componente distribuida entre las distintas clases.

Otra de las clases de este componente es *OperacionesCargarInformacionDatos*, cuyo propósito consiste en seleccionar el tipo de formato de hoja de cálculo del que se van a obtener los datos (*establecerTipoCargarInformacionDatos*), e invocar a las operaciones del mismo.

Para realizar la recuperación de datos de distintos formatos de hoja de cálculo, se hace uso del patrón *Strategy* [Bibliografía-3]. La interfaz de esta

estrategia es *ICargaInformacionDatos*, y está compuesta por las siguientes operaciones:

- *cargarInformacionDatos*: abrir la hoja de cálculo y obtener los datos almacenados en ella.
- *leerDato*: obtener el dato correspondiente de la celda cuyas coordenadas se introducen como parámetro.
- *terminarLectura*: cerrar la hoja de cálculo, liberando todos los recursos que se hayan empleado para su lectura.

Esta interfaz *ICargaInformacionDatos* es implementada por la estrategia *CargaInformacionDatosExcel*, la cual tiene como propósito cargar los datos de una hoja de cálculo de formato Excel 2003. No es necesario incluir más estrategias, ya que teniendo en cuenta el alcance del sistema, únicamente es necesario el formato Excel 2003. Sin embargo, nuevos formatos podrían ser añadidos sin necesidad de realizar otras modificaciones en el componente.

Tal y como se puede apreciar, este componente fomenta la reusabilidad, ya que se podría integrar en cualquier aplicación que necesitase obtener datos de diferentes formatos de hojas de cálculo.

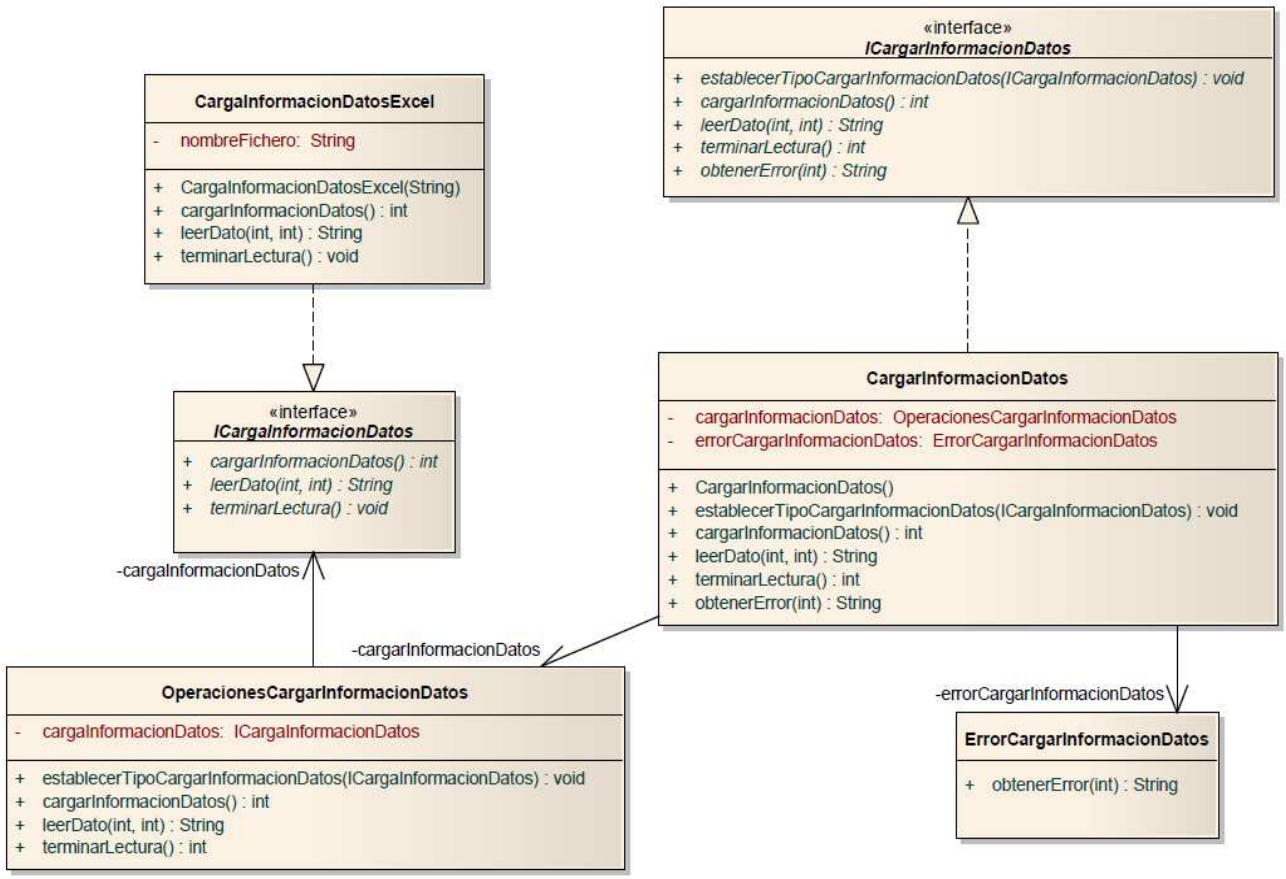

Ilustración 36: diagrama de clases del componente CargarInformacionDatos.

4.1.1.2.7. Diseño del componente Configuracion.

En la Ilustración 37 se presenta el diseño de clases del componente *Configuracion*. Como se puede recordar de la descripción de la arquitectura del sistema, este componente tiene como objetivo proporcionar servicios para el almacenaje o carga de parámetros de configuración de distintos soportes de almacenamiento.

Este componente únicamente dispone de la clase *GestionConfiguracion*, que implementa al puerto del mismo nombre perteneciente a este componente, y cuyo objetivo es englobar los servicios de los componentes internos *EscrituraConfiguracion* y *LecturaConfiguracion*, de manera que los clientes puedan hacer uso de los mismos, sin necesidad de conocer la existencia de dichos componentes.

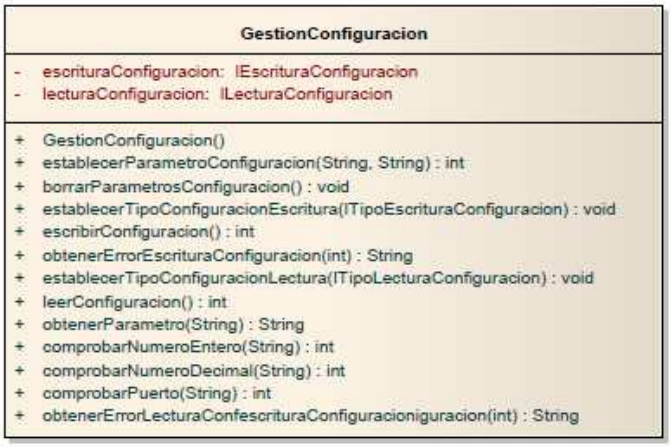

Ilustración 37: diagrama de clases del componente Configuracion.

4.1.1.2.8. Diseño del componente EscrituraConfiguracion.

En la Ilustración 38 se muestra el diagrama de clases del componente *EscrituraConfiguracion*. El objetivo de este componente es ofrecer soporte para el almacenaje de parámetros de configuración en un dispositivo de almacenamiento concreto.

La interfaz que ofrece este componente es *IEscrituraConfiguracion*, la cual es implementada por medio de la clase *EscrituraConfiguracion*, que proporciona una fachada (patrón *Facade* [Bibliografía-3]) para coordinar el acceso a la funcionalidad ofrecida por este componente, repartida entre las distintas clases que lo componen.

En la Ilustración 38 se puede observar también la clase OperacionesEscrituraConfiguracion, cuyo objetivo es la gestión de los distintos parámetros de configuración que se van a guardar en un dispositivo de almacenamiento concreto. Entre las operaciones de esta clase se pueden destacar *establecerParametroConfiguracion*, cuyo propósito debe ser almacenar el nombre y valor de un determinado parámetro de configuración en la tabla *parametrosConfiguracion*, y *escribirConfiguracion*, en la que se debe invocar a la operación *escribirConfiguracion* de *ITipoEscrituraConfiguracion* para guardar los parámetros almacenados en la tabla *parametrosConfiguracion* en un dispositivo de almacenamiento concreto.

Para realizar el almacenamiento en dispositivos de almacenamiento diferentes, se hace uso del patrón *Strategy* [Bibliografía-3]. La interfaz del *Strategy* [Bibliografía-3] es *ITipoEscrituraConfiguracion*, la cual define la operación *escribirConfiguracion* encargada de la implementación de la estrategia de almacenamiento de los parámetros en un soporte determinado. Teniendo en cuenta que el sistema de representación de indicadores obtiene el puerto de escucha de los mensajes de un fichero de configuración, se debe implementar la estrategia *TipoEscrituraConfiguracionFichero*, cuyo objetivo es la obtención de parámetros de configuración de un fichero. A pesar de ello, la inserción de nuevas estrategias no implicaría cambios en el resto del componente.

Finalmente, destacar que este componente puede ser reutilizado en cualquier sistema que necesite de la obtención de parámetros de configuración de distintos soportes de almacenamiento.

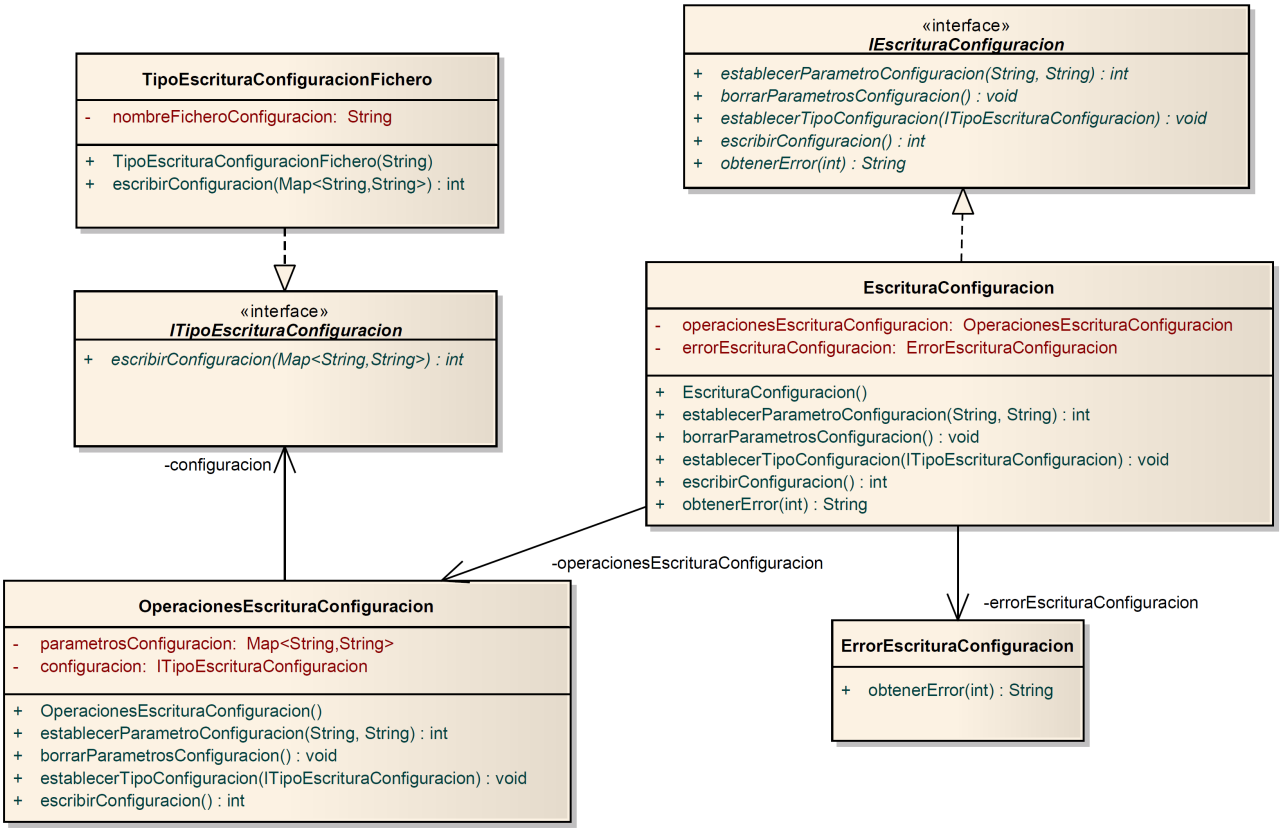

Ilustración 38: diagrama de clases del componente EscrituraConfiguracion.

4.1.1.2.9. Diseño del componente LecturaConfiguracion.

En la Ilustración 39 se muestra el diseño de clases del componente *LecturaConfiguracion*. Como se puede recordar de la descripción de la arquitectura de alto nivel del sistema, este componente tiene como propósito ofrecer servicios para la lectura de parámetros de configuración almacenados en dispositivos diferentes. También proporciona operaciones a los clientes para realizar la comprobación del formato y valor de ciertos parámetros de configuración.

Teniendo en cuenta que el propósito y diseño de este componente es similar al presentado en el apartado 4.1.1.1.9 (con la diferencia de que este debe ser implementado en Java), no se va a profundizar en la descripción del mismo, por motivos de simplicidad.

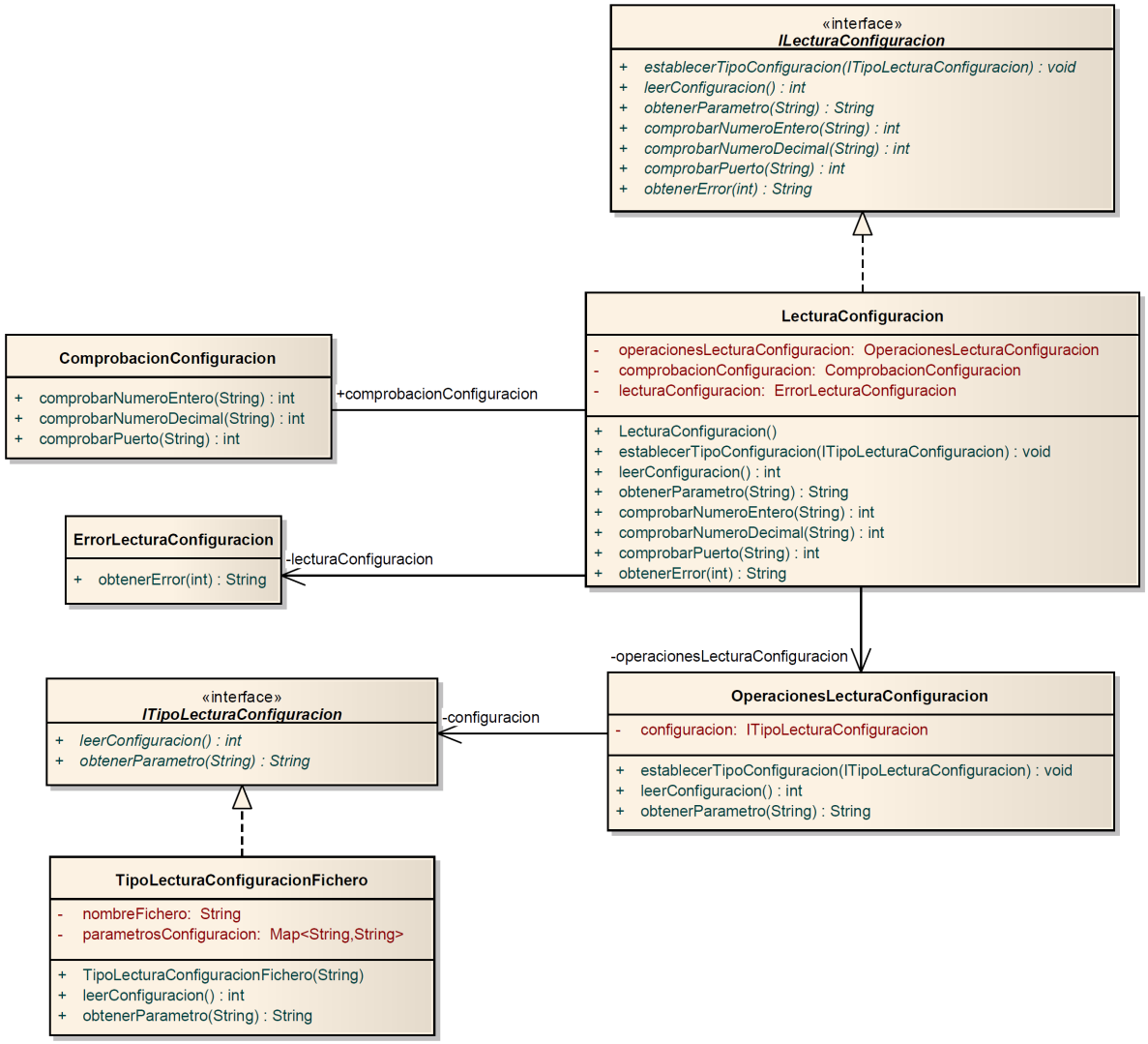

Ilustración 39: diagrama de clases del componente LecturaConfiguracion.

4.1.1.2.10. Diseño del componente Comunicaciones.

El diseño de clases del componente *Comunicaciones* se presenta en la Ilustración 40. Este componente tiene como objetivo abstraer la utilización de distintos mecanismos de comunicación, empleados en la recepción de los mensajes.

Comparando este diagrama de clases con el especificado en el apartado 4.1.1.1.10, se puede apreciar que la idea empleada en el diseño de ambos es muy similar, con la única diferencia de que este componente se centra exclusivamente en la recepción de los mensajes.

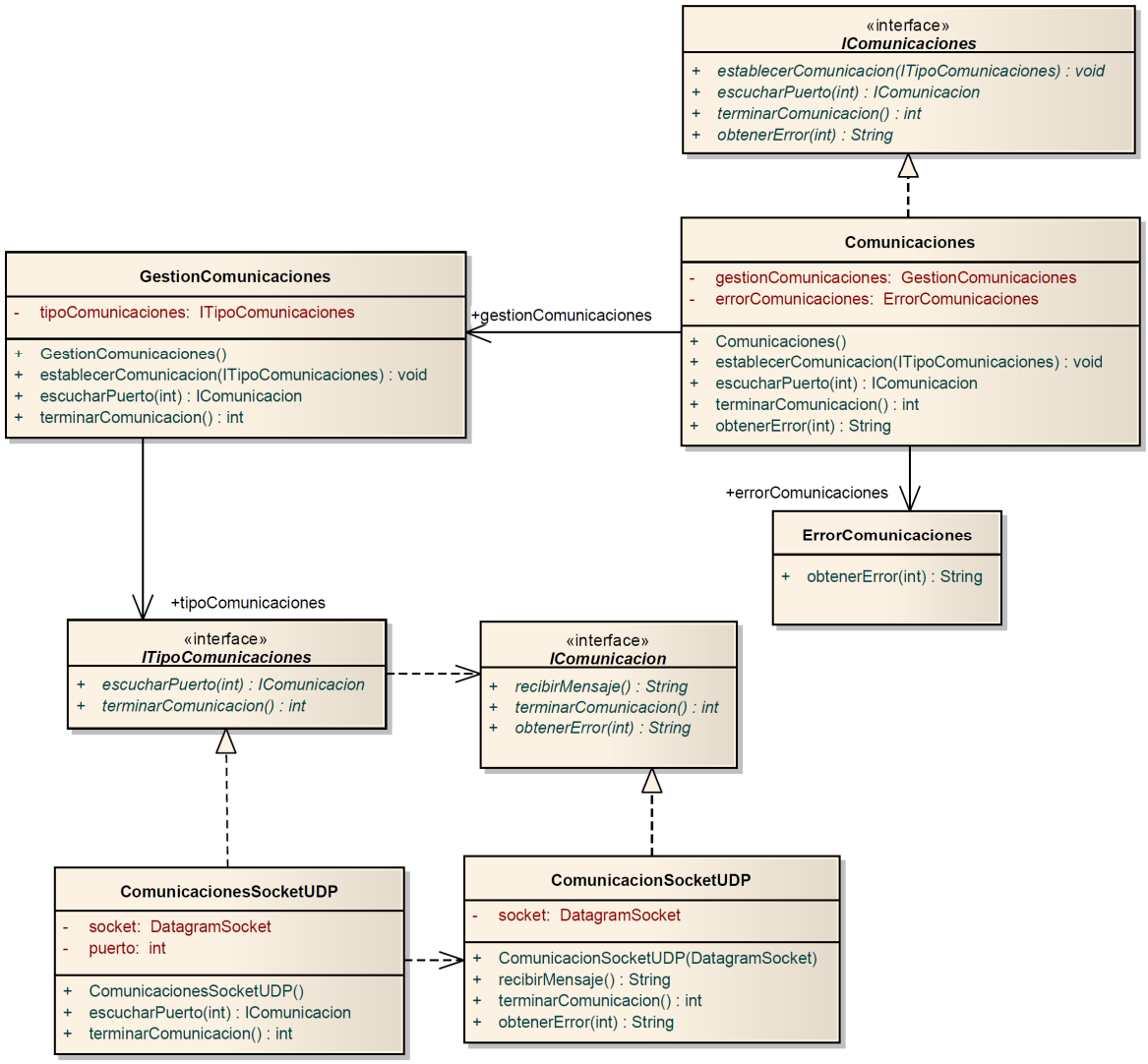

Ilustración 40: diagrama de clases del componente Comunicaciones.

4.1.1.2.11. Diseño del componente ControladorEstadisticas.

En la Ilustración 41 se muestra el diseño de clases del componente *ControladorEstadisticas*. Como ya se especificó en la sección de análisis, este componente se corresponde con el Controlador del modelo MVC [Bibliografía-3], cuyo objetivo es la recepción de los eventos de la interfaz y ejecución de las operaciones apropiadas de *ModeloEstadisticas*, comprobando previamente los parámetros de entrada.

La interfaz que proporciona este componente es *IVCEstadisticas*, cuya implementación es desarrollada por la clase *ControladorEstadisticas*.

Todas las operaciones de esta clase, salvo la operación *obtenerError*, se deben encargan de ejecutar las operaciones del mismo nombre de *ModeloEstadisticas*.

En cuanto a las comprobaciones que debe realizar esta clase, en las operaciones *comenzarAnalisis*, *pararAnalisis*, *continuarAnalisis* y *pausarAnalisis*, se debe verificar el estado en el que se encuentra el sistema

(*estadoAnalizador*), antes de invocar a las mismas (ya que por ejemplo no se debe poder ejecutar *continuarAnalisis* sin antes utilizar *pausarAnalisis*).

En las operaciones *guardarInformacionGraficosPDF*, *guardarInformacionDatosExcel* y *cargarInformacionDatosExcel* se debe comprobar que la extensión de los ficheros es correcta.

En lo que respecta a la operación *establecerPuerto*, se debe comprobar que el puerto introducido como entrada sea de tipo entero y comprendido entre 0 y 65535.

Por último, la operación *obtenerError* de esta clase delega su funcionalidad en la operación de mismo nombre de la clase *ErrorControladorEstadisticas*, y tiene como objetivo proporcionar información de los errores provocados por fallos en las comprobaciones descritas anteriormente. También informa de los errores controlados por el programador, debido a un mal uso de las operaciones que ofrece el componente por parte de los clientes.

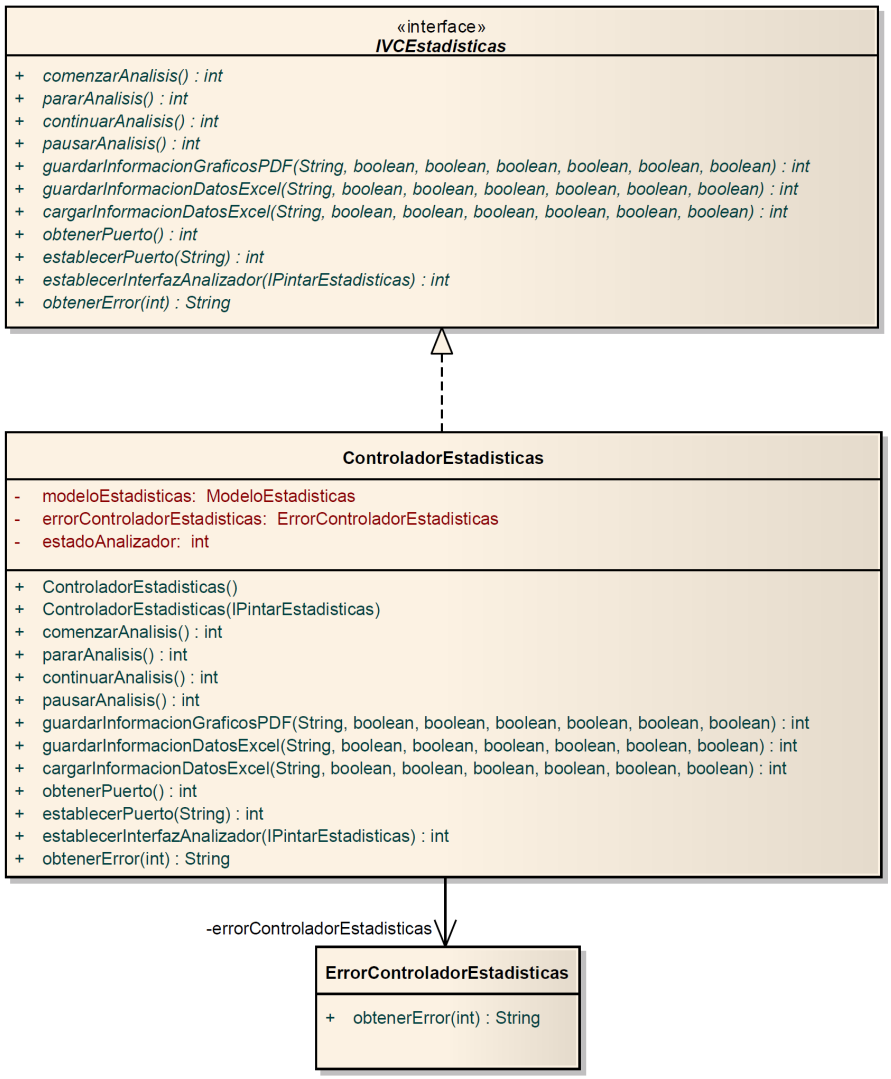

Ilustración 41: diagrama de clases del componente ControladorEstadisticas.

4.1.1.2.12. Diseño del componente VistaEstadisticas.

El diagrama de clases del componente *VistaEstadisticas* se presenta en la Ilustración 42. Como se puede recordar, este componente se corresponde con la Vista del modelo MVC [Bibliografía-3], en la que se implementa la interfaz gráfica del sistema.

Como se puede comprobar, este componente ofrece la interfaz *IPintarEstadisticas*, la cual es utilizada por *ModeloEstadisticas* para enviar el valor de los indicadores de rendimiento a la interfaz gráfica, para su representación. Esta interfaz es implementada por la clase *PintarEstadisticas*. Entre las operaciones de esta clase se puede encontrar *actualizarBytesTransmitidos*, que se debe encargar de invocar a las operaciones *crearGraficoBytesTransmitidos* y *establecerMediaBytesTransmitidos* de la clase *Estadisticas*. Existen otras operaciones para la representación del resto de indicadores como *actualizarTiempoEnvio*, *actualizarTiempoRespuesta*, etc., que siguen el mismo principio que *actualizarBytesTransmitidos*. Otra operación de esta interfaz es *informarFallo*, la cual debe ser utilizada para mostrar mensajes de error por la interfaz gráfica.

La clase *Estadisticas* es la clase encargada de implementar la interfaz gráfica que será utilizada por el usuario para gestionar el sistema, y en la cual se deben representar los indicadores de rendimiento del sistema.

Para la representación de los valores de los indicadores de rendimiento de manera gráfica, en esta clase se deben implementar una serie de operaciones para conseguir dicho propósito. Entre estas operaciones se pueden destacar *crearGraficoBytesTransmitidos*, que se debe encargar de crear la gráfica con los valores del número de bytes transmitidos en cada uno de los mensajes que han sido medidos, y de representar el número de indicadores obtenidos de este tipo, y la operación *establecerMediaBytesTransmitidos*, cuyo objetivo debe ser la representación de la media obtenida de las medidas del número de bytes transmitidos en dichos mensajes. Existen otras operaciones como *crearGraficoTiempoEnvio* y *establecerMediaTiempoEnvio*, *crearGraficoTiempoRespuesta* y *establecerMediaTiempoRespuesta*, etc. que siguen la misma idea que *crearGraficoBytesTransmitidos* y *establecerMediaBytesTransmitidos*, pero para los respectivos indicadores del tiempo de envío y tiempo de respuesta.

En lo que respecta al diseño de la interfaz gráfica, se deja libertad al programador para su realización, pero atendiendo a los siguientes aspectos:

- La interfaz gráfica debe ser creada utilizando Java Swing.
- Las gráficas de los indicadores de rendimiento deben ser creadas utilizando la librería JFreeChart [Referencias-12].
- Se debe mostrar la información según dos modos distintos:
  - Modo básico: mostrar la media del número de bytes transmitidos, tiempo de envío, tiempo de respuesta y tiempo de computación, y las distintas tasas de éxito y fracaso, y aceptación y rechazo. Además, se debe presentar el número total de indicadores obtenidos de cada tipo.

- Modo avanzado: modo básico, más presentación de las diferentes gráficas que muestran los valores de los indicadores de rendimiento.
  - Se deben usar gráficos de tipo lineal y de tipo circular, según lo especificado en los requisitos.

- Debe disponer de un menú, visible en todo momento, dividido en cuatro secciones:
  - Almacenamiento de la información de las estadísticas en formato PDF, y el almacenamiento y carga de los indicadores en formato Excel 2003. También debe permitir reiniciar o salir del sistema.
  - Selección entre el modo básico y avanzado de visualización de las estadísticas.
  - Gestión del sistema mediante las operaciones de comienzo, parada, pausa y continuación.
  - Configuración del puerto por el que el sistema va a recibir los mensajes.
- Se deben tener habilitadas las operaciones que se puedan realizar y deshabilitadas las que no se puedan efectuar en cada instante de la ejecución. En los puntos siguientes, se especifica esta restricción de manera más detallada:
  - Si el sistema se encuentra en estado funcionando, únicamente se pueden tener habilitadas las opciones de parada y pausa, la opción de salida del sistema, y se puede cambiar entre los distintos modos de visualización de las estadísticas.
  - Si el sistema se encuentra en estado pausado, se deben tener habilitadas todas las opciones expuestas en el punto anterior, pero cambiado la opción de pausa por la de continuar.
  - Si el sistema se encuentra parado, se deben poder ejecutar todas las operaciones.
  - Se deben poder seleccionar los indicadores cuyas estadísticas se quieren almacenar en PDF, y los indicadores cuyos valores se desean almacenar u obtener de Excel 2003.

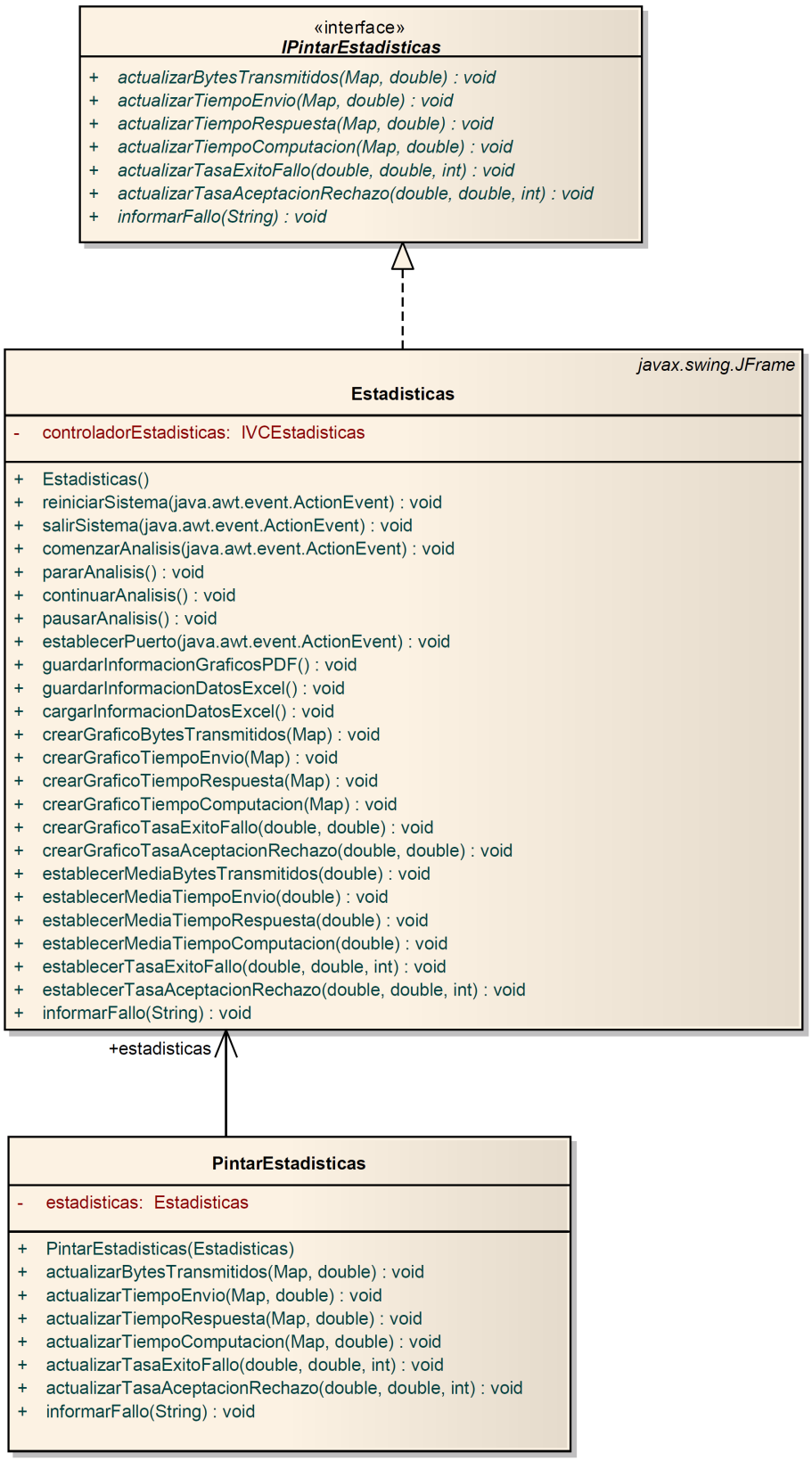

Ilustración 42: diagrama de clases del componente VistaEstadisticas.

4.1.2. Diagramas de secuencia.

En este apartado se van a exponer una serie de diagramas de secuencia, de carácter abstracto, para explicar la secuencia lógica necesaria entre los distintos componentes del sistema, de forma que se pueda explicar la funcionalidad asociada a cada uno de los casos de uso especificados en el apartado 3.7.

El motivo por el que en este apartado se muestra una secuencia lógica entre componentes, es por motivos de simplicidad y claridad, de manera que el lector pueda obtener una idea general del funcionamiento del sistema sin necesidad de especificar detalles más concretos. No obstante, con el objetivo de servir de guía para la posterior implementación, en el documento *MemoriaPFC_DiagramaSecuencia.pdf*, incluido como parte de la documentación de este proyecto, se especifica la interacción concreta entre las clases, por medio de sus operaciones propias (las que fueron comentadas en el apartado 4.1.1).

Como se puede recordar de la especificación de casos de uso, se distinguen dos actores:

- *Usuario NCTUns*: actor encargado de interactuar directamente con NCTUns [Referencias-1], para efectuar las simulaciones del funcionamiento de las distintas entidades de EVIGEN [Bibliografía-1], en distintos escenarios de simulación.
- *Usuario sistema representacion medidas*: actor encargado de reflejar los indicadores de rendimiento en el sistema de representación de los mismos. También se encarga de almacenar los indicadores en formato PDF, y de almacenar y cargar los mismos en formato Excel 2003.

Por tanto, en los subapartados siguientes se presentan los diagramas de secuencia asociados a cada uno de los actores.

4.1.2.1. Diagramas de secuencia asociados a los casos de uso del actor *Usuario NCTUns*.

Como se puede apreciar en la Ilustración 10 del apartado 3.7, el actor *Usuario NCTUns* únicamente realiza el caso de uso *Simular*. Por tanto, en el subapartado siguiente, se presentan los diagramas de secuencia asociados a dicho caso de uso.

4.1.2.1.1. Diagramas de secuencia del caso de uso Simular.

Como se puede recordar, el objetivo de este caso de uso es la simulación del comportamiento de las entidades que participan en EVIGEN [Bibliografía-1] en un escenario concreto.

Teniendo en cuenta que el funcionamiento interno de NCTUns [Referencias-1] para ejecutar la simulación no entra dentro del ámbito de este proyecto (ya que se encuentra implementado), en los diagramas de secuencia presentados en los subapartados posteriores, se muestra la funcionalidad propia asociada a las entidades RSU, DGT y AC (ya que es la parte que se debe implementar en este sistema).

4.1.2.1.1.1. Diagrama de secuencia de simulación global.

En la Ilustración 43 se presenta el diagrama de secuencia asociado a la ejecución de la simulación por parte del actor *Usuario NCTUns*. Es necesario indicar que este diagrama únicamente presenta un carácter lógico, cuyo objetivo es indicar que la funcionalidad a especificar en los apartados siguientes se desarrolla dentro del flujo de ejecución marcado por la simulación (flujo de ejecución *simular*).

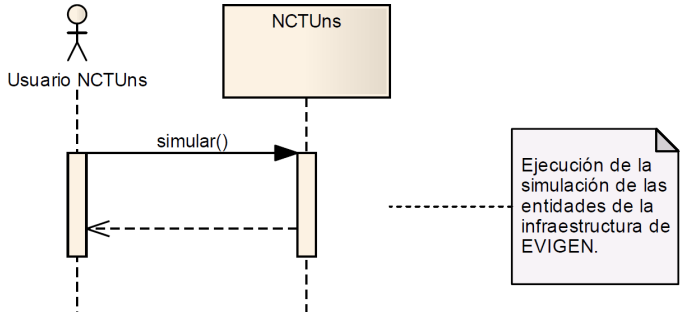

Ilustración 43: diagrama de secuencia de simulación global.

4.1.2.1.1.2. Diagrama de secuencia de arranque de una entidad RSU.

En el diagrama presentado en la Ilustración 44, se presenta la interacción lógica entre componentes que una entidad RSU necesita realizar en primer lugar, para funcionar correctamente.

En primer lugar, se debe obtener del fichero de configuración de la RSU los parámetros de la misma (*obtenerConfiguracion*).

Una vez obtenidas estas variables, se debe especificar el fichero en el que se ubicará el log, así como el identificador que será utilizado por la RSU para identificar las entradas que efectúe la misma (*establecerParametrosLog*).

Posteriormente, se debe indicar la ruta en la que se sitúa el certificado de la entidad, la ruta del certificado de la autoridad de certificación que lo emite, y el fichero en el que se encuentra su clave privada (*establecerParametrosCriptografia*).

Finalmente, se deben establecer el resto de parámetros que necesita la entidad para su funcionamiento, y por último, se deben crear los hilos de ejecución independientes encargados de la recepción de los mensajes enviados por el resto de entidades (*iniciarRecepcionMensajes)*.

Es necesario indicar que no se van a especificar los diagramas de secuencia de las entidades DGT y AC, debido a que su secuencia lógica es similar a la presentada en el diagrama de la Ilustración 44.

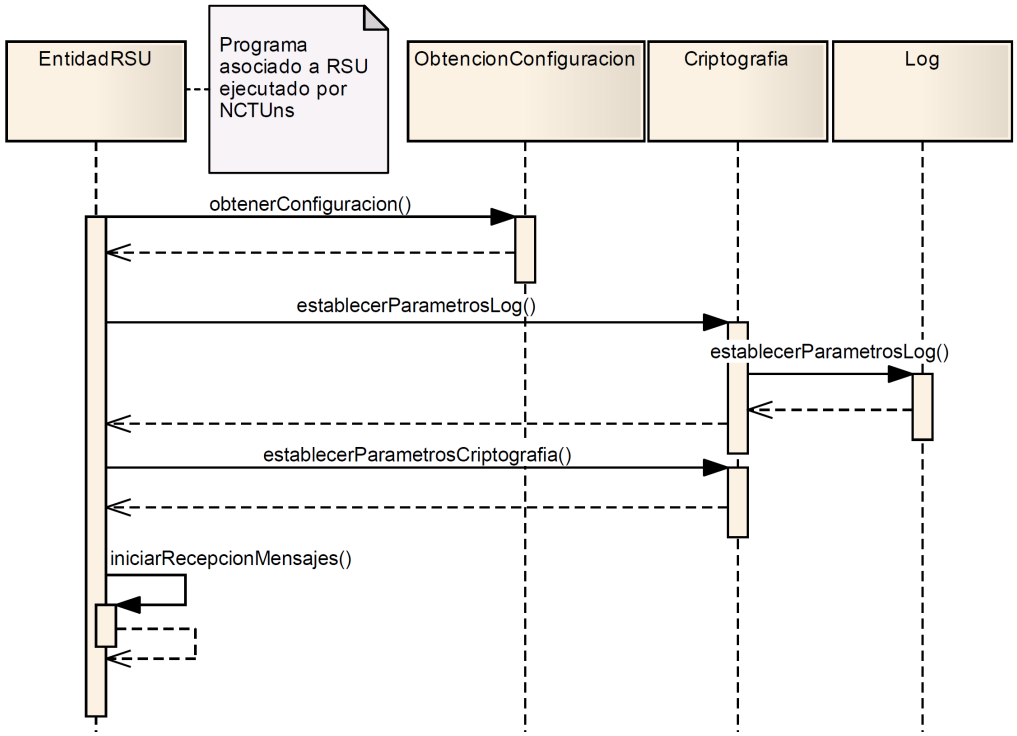

Ilustración 44: diagrama de secuencia de arranque de entidad RSU.

4.1.2.1.1.3. Diagrama de secuencia de determinación de la cercanía de una RSU con la DGT.

En la Ilustración 45 se presenta el diagrama de secuencia lógico correspondiente con la funcionalidad del sistema encargada de determinar si una RSU se encuentra cercana a la DGT por una distancia de comunicación.

Para ello, como se describió en la especificación del diseño de clases, la DGT envía un mensaje con su estado a las RSU. Cuando una RSU recibe este mensaje (*recibirMensaje*), se debe procesar el mismo para extraer y decodificar su contenido *(procesarMensajeEstado*).

Finalmente, se debe obtener la distancia de la RSU con la DGT *(calcularDistancia*) y verificar si es menor o igual que la distancia máxima de comunicación permitida.

Resulta necesario indicar que para determinar las RSU cercanas a otra RSU, la secuencia lógica empleada entre componentes es similar a la mostrada en la Ilustración 45, por lo que por motivos de simplicidad no se va a especificar el correspondiente diagrama.

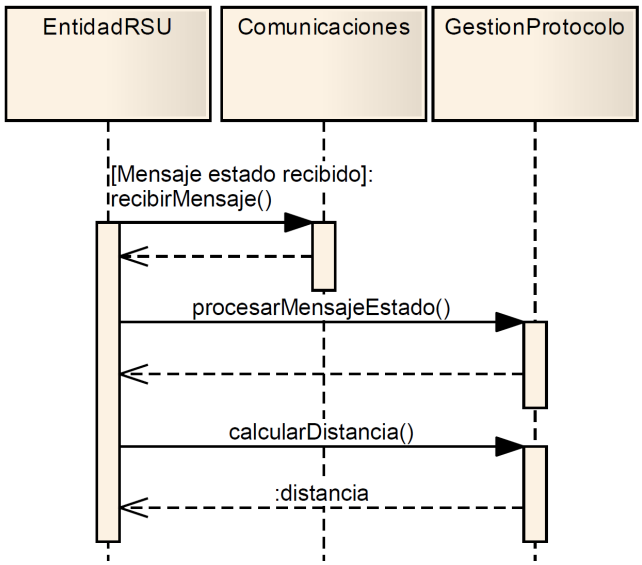

Ilustración 45: diagrama de secuencia de determinación de cercanía RSU-DGT.

4.1.2.1.1.4. Diagrama de secuencia de detección de un vehículo infractor.

En el diagrama de secuencia expuesto en la Ilustración 46, se presenta la secuencia lógica que se necesita efectuar para determinar la cercanía de una RSU con un determinado vehículo, detección de una infracción, y envío de un mensaje con una notificación de infracción a la DGT.

En primer lugar, la RSU debe recibir el *beacon* enviado desde los vehículos para determinar si se encuentra lo suficiente cerca de los mismos como para comunicarse con ellos *(recibirMensaje*).

Recibido el *beacon*, se debe procesar el mismo para decodificar su contenido y comprobar que éste es correcto (*procesarBeaconCoche*), y posteriormente, se debe obtener la distancia entre la RSU y el vehículo emisor del *beacon* (*calcularDistancia*) para determinar si se encuentran lo suficientemente próximos para comunicarse.

En caso de que se encuentren cercanos, se debe verificar si el vehículo ha cometido alguna infracción (*comprobarInfraccion*), lo cual, en caso de ser así, implica que se deba crear una notificación de infracción y un mensaje que la contenga para enviarlo a la DGT *(crearMensajeNotificacionInfraccion*). Para crear el mensaje, se debe obtener en primer lugar un identificador único del mismo para evitar ataques de repetición. Además, teniendo en cuenta que se tiene que garantizar la integridad y no repudio del contenido del mensaje con la notificación de infracción, se debe firmar el mismo (*firmar*), lo cual implica la inserción de una nueva entrada en el log (*escribirEnLog*).

También se debe considerar, tal y como se puede apreciar en la Ilustración 46, la medición del tiempo de computación de la firma realizada (*comenzarMedirTiempoComputacion* y *terminarMedirTiempoComputacion*)

Como se puede comprobar, al igual que se hace en el resto de diagramas posteriores, el tiempo de computación también mide el tiempo específico de la codificación o decodificación, pero considerando que este tiempo es prácticamente despreciable frente al tiempo empleado en la ejecución de la

operaciones criptográficas, y siendo fiel al diseño realizado, se efectúan las mediciones desde cada una de las distintas entidades.

Finalmente, se debe enviar el mensaje con la notificación de infracción a la DGT (*enviarMensaje*). Además, teniendo en cuenta el envío realizado, se debe medir el número de bytes transmitidos (*medirBytesTransmitidos*) y el tiempo de envío del mensaje (*comenzarMedirTiempoEnvio* y *terminarMedirTiempoEnvio*). Todo esto se realiza dentro de la operación lógica *enviarNotificacionInfraccion*.

Resulta necesario añadir que, en caso de que la RSU no se encuentre próxima a la DGT, y el mensaje tenga que ser enviado al resto de RSU cercanas, la secuencia es la misma que la mostrada en este diagrama, teniendo en cuenta que se debe efectuar el envío a cada una de las RSU, realizando las correspondientes mediciones de número de bytes transmitidos y tiempo de envío por cada una de ellas.

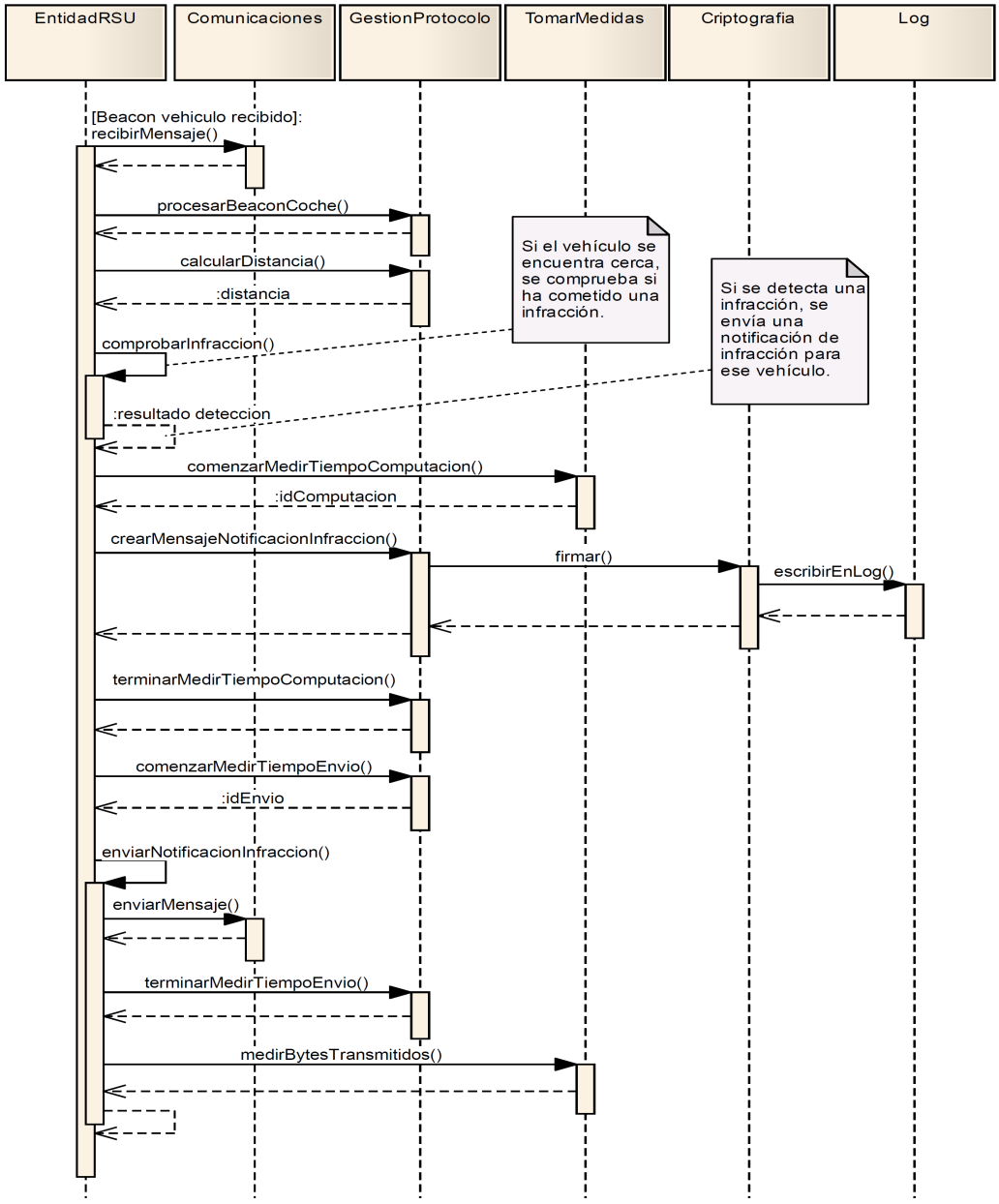

Ilustración 46: diagrama de secuencia de detección de un vehículo infractor.

4.1.2.1.1.5. Diagrama de secuencia del reenvío de una notificación de infracción.

En el diagrama mostrado en la Ilustración 47, se describe la interacción lógica necesaria entre los distintos componentes, para desarrollar la funcionalidad de reenvío desde una RSU a la DGT, o al resto de RSU cercanas a la misma en caso de no ubicarse próxima a la DGT, de un mensaje con una notificación de infracción recibido de otra RSU.

Por ello, en primer lugar, se debe obtener el mensaje con la notificación de infracción *(recibirMensaje*), y se debe procesar el mismo para decodificar su contenido y verificar que éste es correcto *(procesarMensajeNotificacionInfraccionParcial*), y que no ha sido recibido previamente.

Finalmente, se debe reenviar el mismo a la DGT si se encuentra cercana a la RSU según la distancia de comunicación determinada, o al resto de RSU en caso contrario (misma operación *enviarNotificacionInfraccion* vista en el apartado 4.1.2.1.1.5).

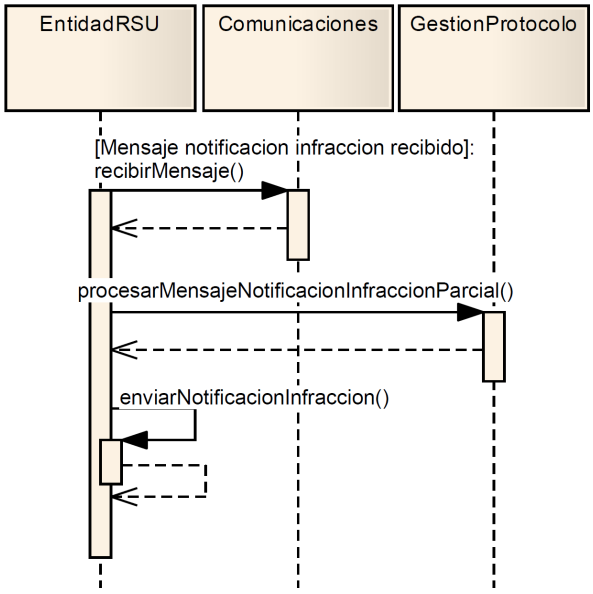

Ilustración 47: diagrama de secuencia de reenvío de notificación de infracción.

4.1.2.1.1.6. Diagrama de secuencia de reenvío de mensaje con una notificación de sanción.

En el diagrama mostrado en la Ilustración 48: diagrama de secuencia de reenvío de una notificación de sanción.Ilustración 48 se observa la interacción lógica necesaria entre los componentes para efectuar el envío de un mensaje con una notificación de sanción recibido en una RSU, al vehículo infractor.

En primer lugar, se debe obtener el mensaje recibido (*recibirMensaje*), y posteriormente, se debe procesar el mismo para decodificar su contenido y comprobar que su formato y campos son correctos, que no ha sido entregado previamente al vehículo infractor, y que el tiempo de vida de éste no ha expirado (*procesarMensajeNotificacionSancion*).

Una vez comprobado que el mensaje es correcto, se debe decrementar el tiempo de vida del mismo en una unidad, y posteriormente enviar el mensaje al vehículo infractor (*enviarMensaje*). Al igual que se ha visto en casos anteriores, se debe medir el número de bytes transmitidos y el tiempo empleado en el envío del mensaje con la notificación de sanción.

Finalmente, con el fin de mostrar el envío de la notificación de sanción al vehículo infractor por el sistema de visualización de mensajes, se debe ejecutar la operación *enviadaNotificacion* proporcionada por la interfaz *IEnlaceComponente*.

Resulta necesario indicar que el mensaje con la notificación de sanción únicamente puede ser transmitido al vehículo infractor si se encuentra circulando próximo a la RSU, siendo en caso contrario enviado al resto de RSU cercanas. Por tanto, considerando que la secuencia para desarrollar dicha funcionalidad es bastante similar a la presente, no va a ser presentada.

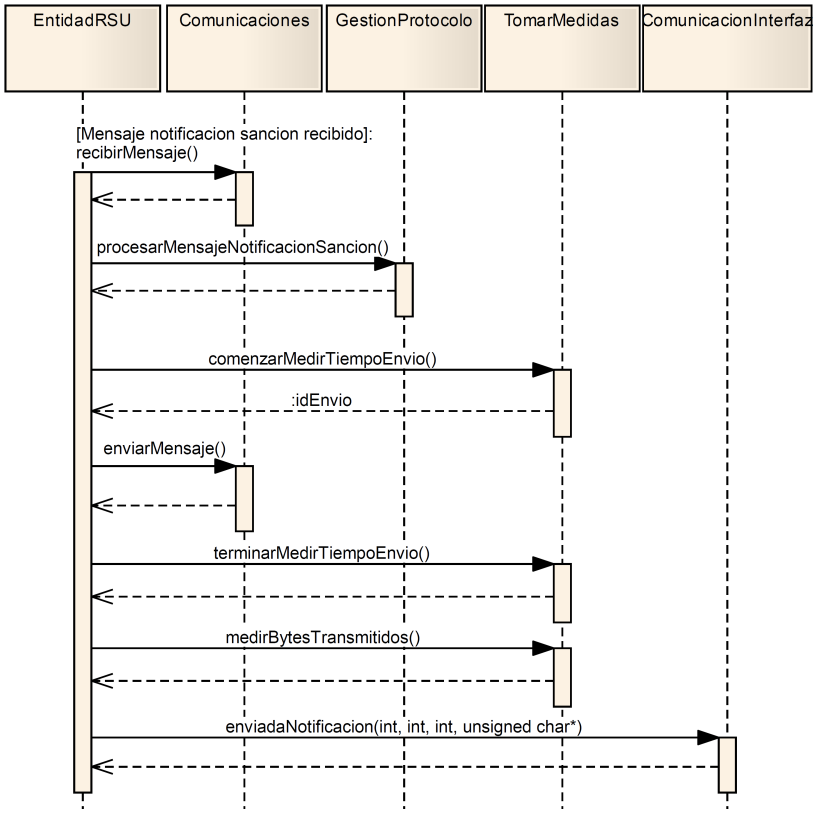

Ilustración 48: diagrama de secuencia de reenvío de una notificación de sanción.

4.1.2.1.1.7. Diagrama de secuencia de reenvío del mensaje con la CRL.

En el diagrama presentado en la Ilustración 49, se presenta la secuencia lógica necesaria para desarrollar la funcionalidad del reenvío de una RSU del mensaje que contiene la CRL, al resto de RSU y vehículos cercanos a la misma.

En primer lugar, se debe obtener el mensaje con la CRL (*recibirMensaje*), para posteriormente, procesar el mismo para decodificar su contenido y comprobar que éste sea correcto, y que el tiempo de vida del mensaje no haya expirado (*procesarMensajeCRL*).

Una vez se han realizado las comprobaciones, se debe decrementar el tiempo de vida del mensaje, y se debe enviar el mismo a los vehículos (*enviarCRL_A_Coches*) y al resto de RSU (*enviarCRL_A_RSUs*) próximas a la RSU por una distancia de comunicación, realizando como se puede verificar, las correspondientes medidas de número de bytes transmitidos y tiempo de envío.

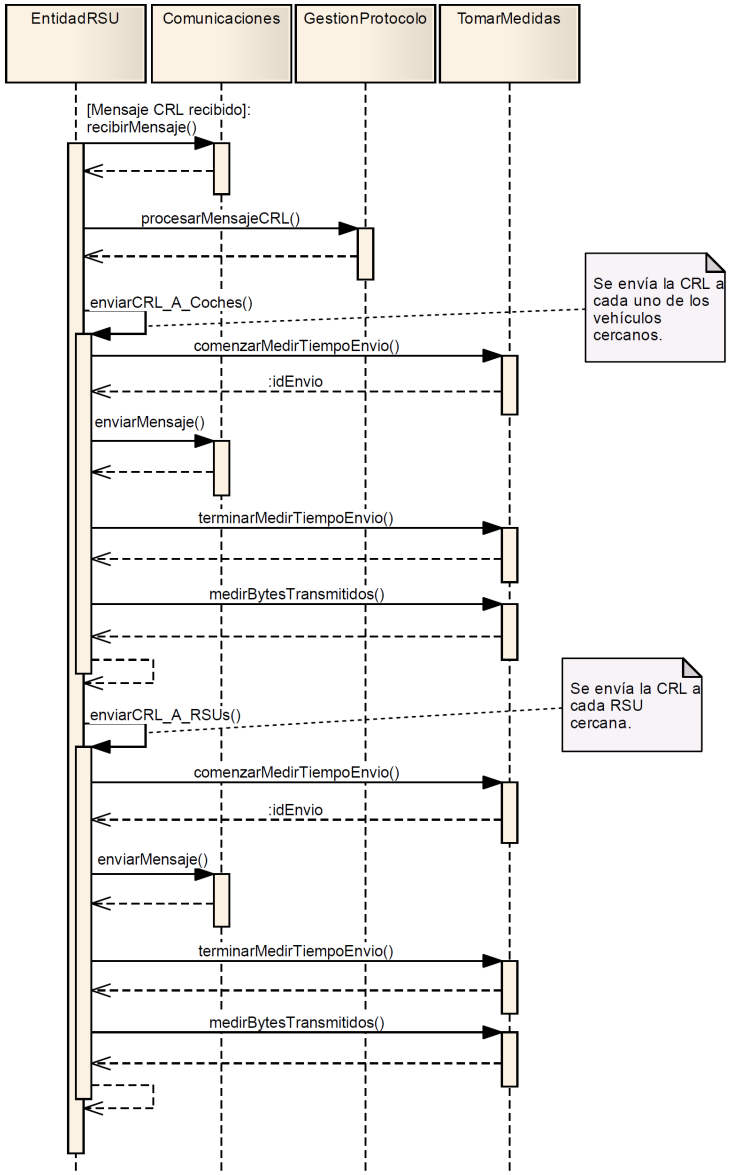

Ilustración 49: diagrama de secuencia de reenvío del mensaje con la CRL.

4.1.2.1.1.8. Diagrama de secuencia de envío de mensaje con una evidencia.

En el diagrama mostrado en la Ilustración 50 se muestra la secuencia lógica necesaria para la realización de la funcionalidad de reenvío desde una RSU a la DGT, de un mensaje con una evidencia de respuesta a una notificación de sanción.

Al igual que en casos anteriores, se debe obtener el mensaje enviado (*recibirMensaje*), y se debe extraer y decodificar el contenido del mismo para verificar que su formato y campos (no cifrados) son correctos, y que no ha sido recibido en dicha RSU con anterioridad.

Una vez se ha verificado que la evidencia está libre de errores, se debe utilizar la operación *recibidaEvidencia* de *ComunicacionInterfaz*, para indicar al sistema de visualización de mensajes, desarrollado en el Proyecto paralelo [Bibliografía-2], de la recepción del mensaje con la evidencia procedente del vehículo infractor que la ha creado.

Finalmente, se debe reenviar el mensaje a la DGT (*enviarMensaje*), realizando las correspondientes medidas del número de bytes transmitidos y tiempo de envío.

Resulta necesario indicar que si la RSU no se encuentra próxima a la DGT por una distancia de comunicación, el envío y mediciones se tendrían que realizar por cada una del resto de RSU cercanas, y no se tendría que hacer uso de la operación *recibidaEvidencia*, ya que el sistema de visualización de mensajes no muestra la interacción entre RSU.

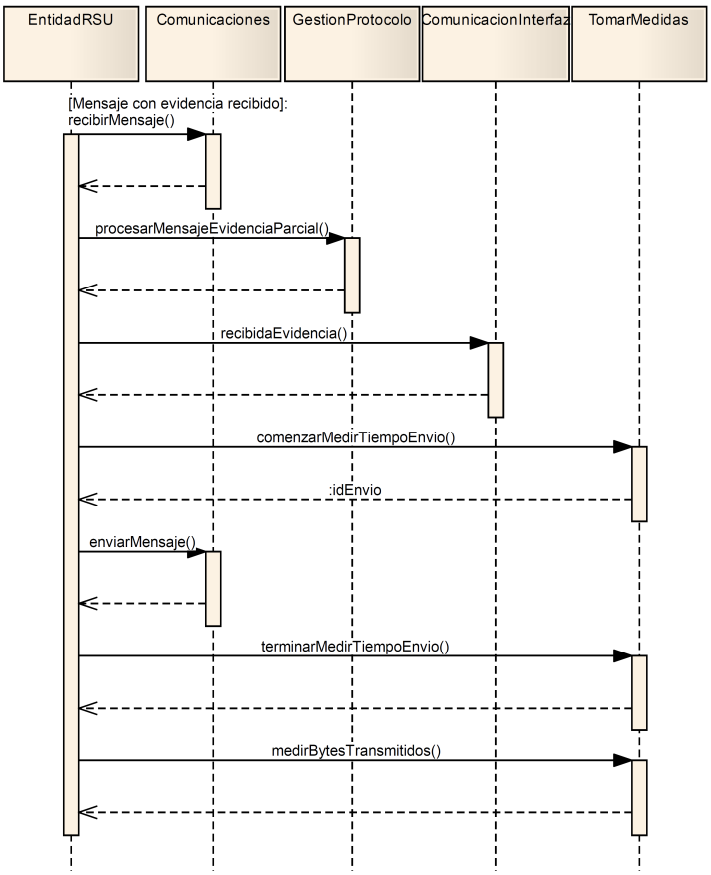

Ilustración 50: diagrama de secuencia de envío de mensaje con una evidencia.

4.1.2.1.1.9. Diagrama de secuencia de envío de *beacon* de una RSU.

En el diagrama presentado en la Ilustración 51, se presenta la interacción lógica necesaria para efectuar el envío periódico del *beacon* de una RSU a los vehículos que se encuentran circulando cercanos a ello (aunque este *beacon* también es utilizado por la DGT, AC y resto de RSU para conocer la ubicación de la RSU concreta).

En primer lugar, se debe obtener la información de estado (identificador y localización) propia de la RSU necesaria para la creación del *beacon*, la cual procede de la simulación concreta.

Una vez obtenida, se debe codificar dicha información con el objetivo de transmitirla al resto de entidades por *broadcast* de manera periódica. Además, como se puede apreciar en la Ilustración 51, se debe hacer uso de la operación *procesarInfoEstadoRSU* con el fin de notificar al sistema de visualización de mensajes del envío del *beacon*.

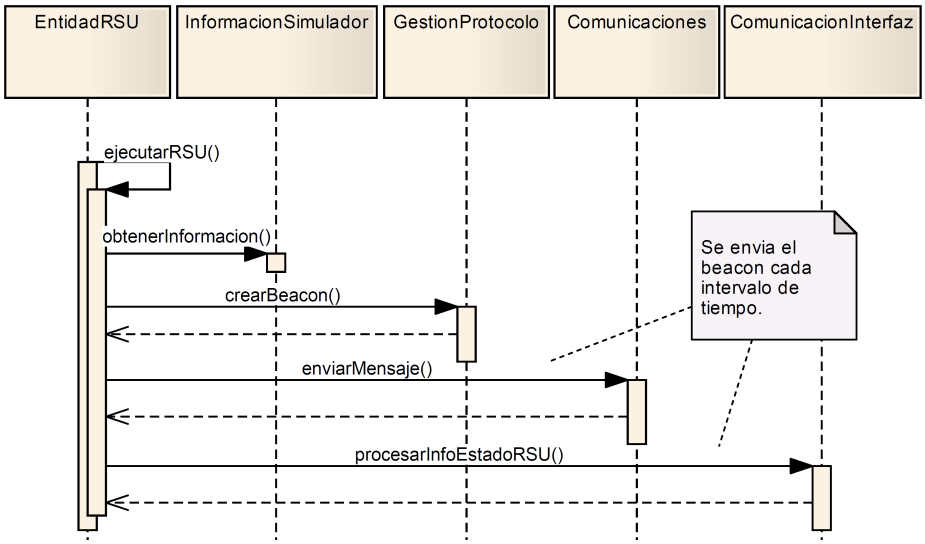

Ilustración 51: diagrama de secuencia de envío de *beacon* de una RSU.

4.1.2.1.1.10. Diagrama de secuencia de envío de IP de AC.

Como se puede recordar, cuando la DGT quiere mandar el identificador de un vehículo cuyo certificado desea revocar, como no posee la IP de la AC desde el inicio de su ejecución, le envía un mensaje de petición a las RSU para que éstas la respondan con dicha IP. En el diagrama mostrado en la Ilustración 52, se presenta la secuencia lógica desde que una RSU recibe el mensaje de petición, hasta que envía la IP concreta a la DGT.

Como se puede verificar, una vez se recibe la petición (*recibirMensaje*), si la RSU conoce la IP de la AC, debe codificar la misma (*codificarIP*) y enviarla a la DGT (*enviarMensaje*).

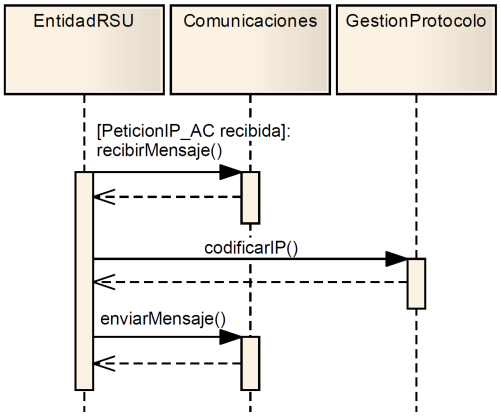

Ilustración 52: diagrama de secuencia de envío de IP a AC.

4.1.2.1.1.11. Diagrama de secuencia de envío de mensaje con un testimonio.

En la Ilustración 53 se presenta el diagrama de secuencia que se corresponde con la funcionalidad de reenvío desde una RSU, la cual

desempeña también el papel de un vehículo no equipado, al vehículo infractor, de un mensaje con el testimonio de un testigo determinado,.

En primer lugar, se debe obtener el mensaje recibido (*recibirMensaje*), y se debe procesar el mismo con el fin de extraer el testimonio cifrado y el identificador del vehículo infractor al que va dirigido, comprobando además, que dicho identificador es válido (*procesarTestimonio*) y que el tiempo de vida del mensaje no ha expirado.

Verificado que el mensaje con el testimonio es correcto, se debe hacer uso de la operación *recibidoMensajeNE_RSU*, para enviar al sistema de visualización de mensajes la información relacionada con la recepción del mismo.

Posteriormente, se debe decrementar en una unidad el tiempo de vida del mensaje, y se debe enviar al vehículo infractor (*enviarMensaje*), efectuando la medición correspondiente con el número de bytes transmitidos y el tiempo de envío.

Finalmente, con el fin de transmitir al sistema de visualización de mensajes la información del mensaje con el testimonio enviado al vehículo infractor, se debe utilizar la operación *enviadoMensajeDesdeNE_RSU*.

En el diagrama presentado, el testimonio es enviado al vehículo infractor al que va dirigido. Sin embargo, si dicho vehículo no se encuentra circulando cerca de la RSU, el envío se debe realizar a las RSU y vehículos no equipados próximos a ella. Por tanto, considerando que la secuencia asociada a dicho caso es similar a la presente, no se va a incluir el correspondiente diagrama, siendo los cambios con respecto a éste, que los envíos se realizan al resto de RSU y vehículos no equipados cercanos, con sus correspondientes indicadores, y que se debe ejecutar la operación *enviadoMensajeDesdeNE_RSU* por cada transmisión a un vehículo no equipado.

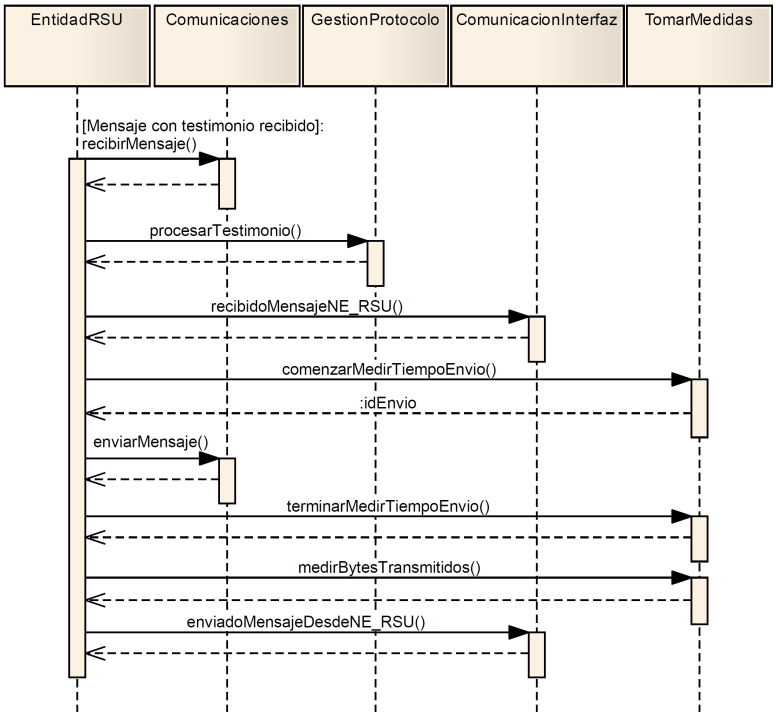

Ilustración 53: diagrama de secuencia de reenvío de mensaje con un testimonio.

4.1.2.1.1.12. Diagrama de secuencia de determinación de RSU próximas a DGT.

Como se puede recordar, para que la DGT pueda determinar si se encuentra lo suficientemente cerca de una RSU para comunicarse con ella, debe recibir el *beacon* enviado por ella y determinar si la distancia entre ambas es menor o igual a la distancia máxima de comunicación establecida.

Considerando que el objetivo de esta funcionalidad es similar a la del apartado 4.1.2.1.1.3, por motivos de simplicidad, en este caso no se va a incluir el correspondiente diagrama de secuencia.

4.1.2.1.1.13. Diagrama de secuencia de envío de mensaje con una notificación de sanción.

En el diagrama mostrado en la Ilustración 54, se presenta la interacción lógica necesaria entre componentes, para realizar la funcionalidad asociada con la recepción del mensaje con una notificación de infracción, creación de la notificación de sanción correspondiente, y transmisión de ésta a las RSU vecinas, con el fin de que éstas se la reenvíen al vehículo infractor.

De la misma forma que se ha visto en casos anteriores, cuando la DGT recibe el mensaje con la notificación de infracción, debe obtener el mismo (*recibirMensaje*) y procesarlo con el fin de decodificar su contenido, y comprobar que su formato y campos son correctos, y que no ha sido recibido anteriormente en la entidad (*procesarMensajeNotificacionInfraccion*). También se debe verificar que el vehículo que ha cometido la infracción no ha sido sancionado con anterioridad dentro de un intervalo de tiempo concreto.

Considerando que se debe garantizar la autenticación del emisor e integridad y no repudio del mensaje con la notificación de infracción, tal y como se observa en el diagrama, se debe comprobar la validez del certificado de la RSU (*comprobarCertificado*) y verificar que la firma ha sido realizada por ella. Teniendo en cuenta que las operaciones de comprobación de certificado y firma son criptográficas, se debe incluir una entrada en el log por cada una de ellas (*escribirEnLog*), y además, se debe medir el respectivo tiempo de computación.

Realizadas todas las comprobaciones sobre el mensaje con la notificación de infracción, se debe crear la notificación de sanción correspondiente (*crearNuevaNotificacion*), para posteriormente enviarla contenida en un mensaje a las RSU cercanas a la DGT (*enviarNotificacion_A_RSU*).

Para ello, en primer lugar se debe codificar el contenido de la notificación de sanción, con el fin de introducirlo en el mensaje a enviar a las RSU (*crearMensajeNotificacionSancion*). Este mensaje debe tener un identificador único generado de manera aleatoria, el cual, junto con el contenido de la notificación de sanción, debe ser firmado (*firmar*) con la clave privada de la DGT. Además, considerando la realización de la firma, se debe medir su correspondiente tiempo de computación. Por último, se le debe adjuntar a dicho mensaje un tiempo de vida máximo, con el objetivo de impedir su circulación indefinida por la red.

Una vez creado el mensaje, se debe transmitir el mismo a las RSU próximas a la DGT por una distancia de comunicación, y además, por cada

envío, se deben realizar las correspondientes mediciones del número de bytes transmitidos y tiempo de envío.

Finalmente, teniendo en cuenta que la notificación de sanción transmitida debe ser respondida con una evidencia destinada a recurrirla, tras la realización del envío, se debe comenzar la medición del tiempo de respuesta empleado en la recepción de dicha evidencia.

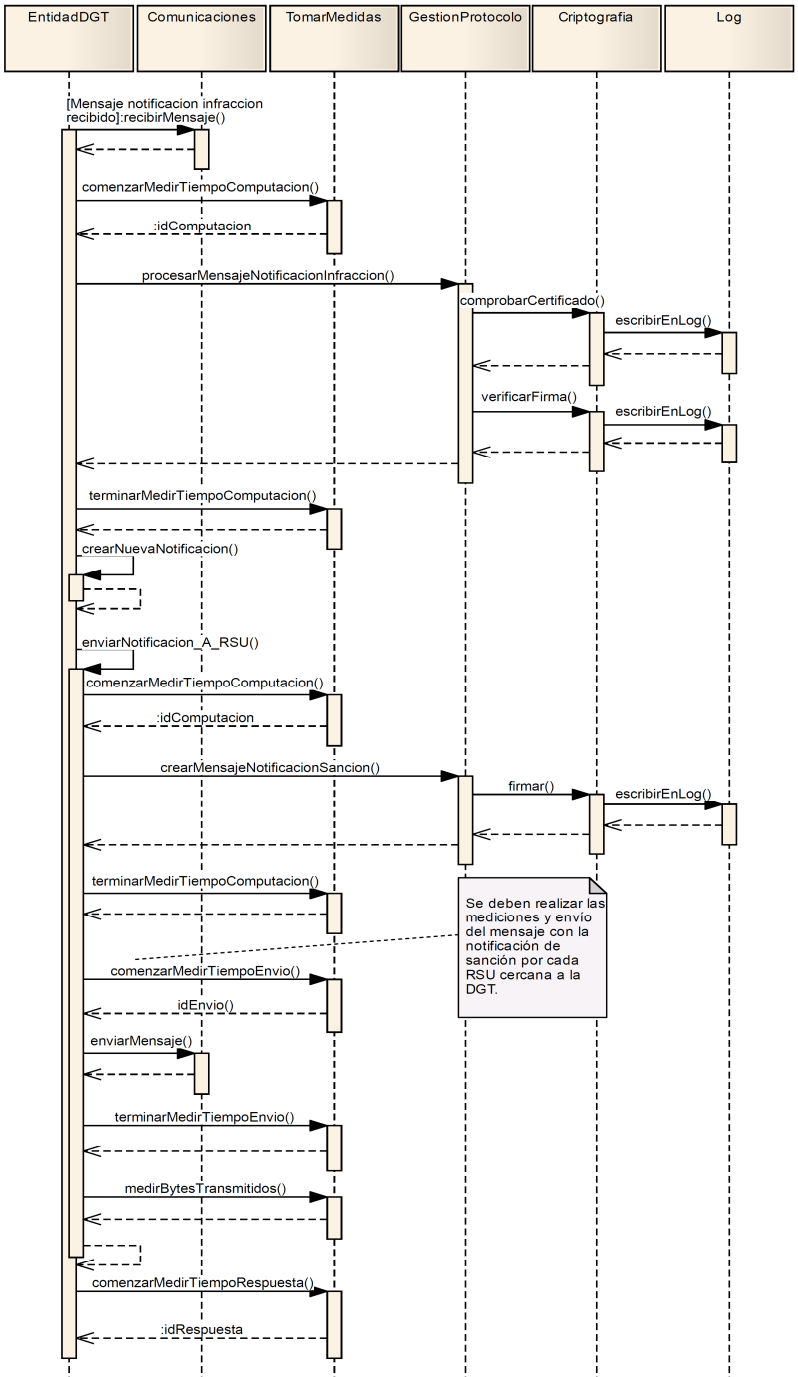

Ilustración 54: diagrama de secuencia de envío de notificación de sanción.

4.1.2.1.1.14. Diagrama de secuencia de reenvío de una notificación de sanción por la DGT.

En el diagrama presentado en la Ilustración 55, se muestra la secuencia lógica necesaria entre los componentes para realizar la funcionalidad asociada

al reenvío de una notificación de sanción, tras la expiración del tiempo destinado a la espera de la evidencia de respuesta a la misma.

La operación *ejecutarGestionNotificaciones* es la encargada de la realización de los reenvíos de las notificaciones de sanción. Por tanto, si expira el tiempo dedicado a la recepción de la evidencia de respuesta a una notificación, se debe reenviar la misma (operación *enviarNotificacion_A_RSU* del apartado Ilustración 544.1.2.1.1.13) y se debe indicar al sistema de representación de indicadores que la finalización del protocolo no ha tenido éxito (*protocoloFinalizadoErroneamente*).

Finalmente, resulta imprescindible destacar que únicamente se ha especificado el caso en el que el número de reenvíos permitidos no ha sido superado, ya que en caso de producirse, se tendría que efectuar el envío de un mensaje de revocación de certificado a la AC (esta funcionalidad se encuentra descrita dentro del apartado 4.1.2.1.1.15).

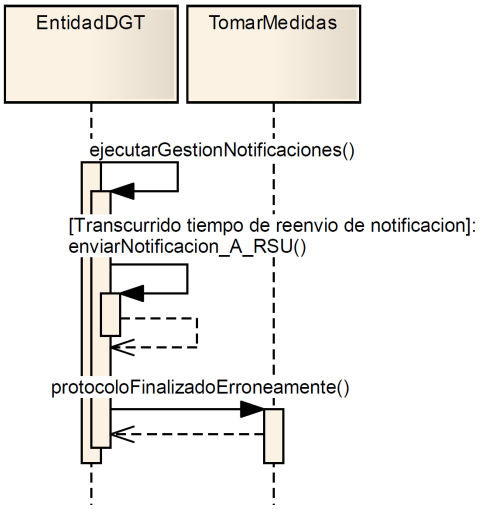

Ilustración 55: diagrama de secuencia de reenvío de notificación de sanción por DGT.

4.1.2.1.1.15. Diagrama de secuencia de comprobación de una evidencia.

Los diagramas mostrados en las páginas siguientes presentan la interacción lógica necesaria entre los distintos componentes, para realizar la funcionalidad relacionada con la recepción del mensaje con una evidencia destinada a recurrir una determinada sanción y comprobación de la misma. Además, en este caso se muestra también la secuencia que se debe seguir para efectuar el envío del mensaje de revocación de certificado a la AC con el identificador de un vehículo concreto, lo cual se produce si el dato de consenso recibido en la evidencia no se corresponde con los testimonios aportados por los testigos.

Como se puede apreciar en la Ilustración 56: diagrama de secuencia 1 de comprobación de una evidencia.Ilustración 56, en primer lugar se debe obtener el mensaje con la evidencia recibido (*recibirMensaje*), y procesar el mismo con el objetivo de decodificar y descifrar su contenido, y verificar que el mismo es correcto. Para ello, primeramente se debe asegurar que el mensaje con la evidencia no ha sido recibido previamente y, que la evidencia contenida en el mismo se corresponde con una notificación válida. Después, se debe

comprobar que el identificador de la notificación y dato de consenso contenidos son correctos. Posteriormente, se debe verificar que el certificado del vehículo infractor adjunto en el mensaje es correcto *(comprobarCertificado*), y se debe utilizar el mismo para verificar la validez de la firma sobre el dato de consenso (*verificarFirma*). Después, por cada ticket y condiciones del testimonio se debe, en primer lugar, descifrar el ticket con la clave privada de la DGT (*descifrar*), para posteriormente, verificar el certificado del testigo (*comprobarCertificado*) y su firma efectuada sobre la clave AES contenida en el mismo (*verificarFirma*), con el fin de asegurar que las condiciones de testimonio cifradas con dicha clave proceden del testigo. Finalmente, se debe hacer uso de dicha clave para descifrar las condiciones del testimonio por medio del algoritmo AES-CCM (*descifrarAES*), y se debe comprobar que el contenido de las mismas es correcto. Resulta necesario resaltar que debido a la realización de diversas operaciones criptográficas, se deben incluir sus entradas correspondientes en el log (*escribirEnLog*), y se debe medir el tiempo de computación empleado en su realización.

Una vez se tiene la evidencia totalmente decodificada, se debe concluir la medición del tiempo empleado en esperar la recepción de la evidencia (*terminarMedirTiempoRespuesta*), y se debe señalar que el protocolo ha finalizado correctamente (*protocoloFinalizadoCorrectamente*).

Posteriormente, se debe utilizar la operación *comprobarEvidencia* para verificar que el dato de consenso incluido en la evidencia se corresponde con el testimonio aportado por los testigos.

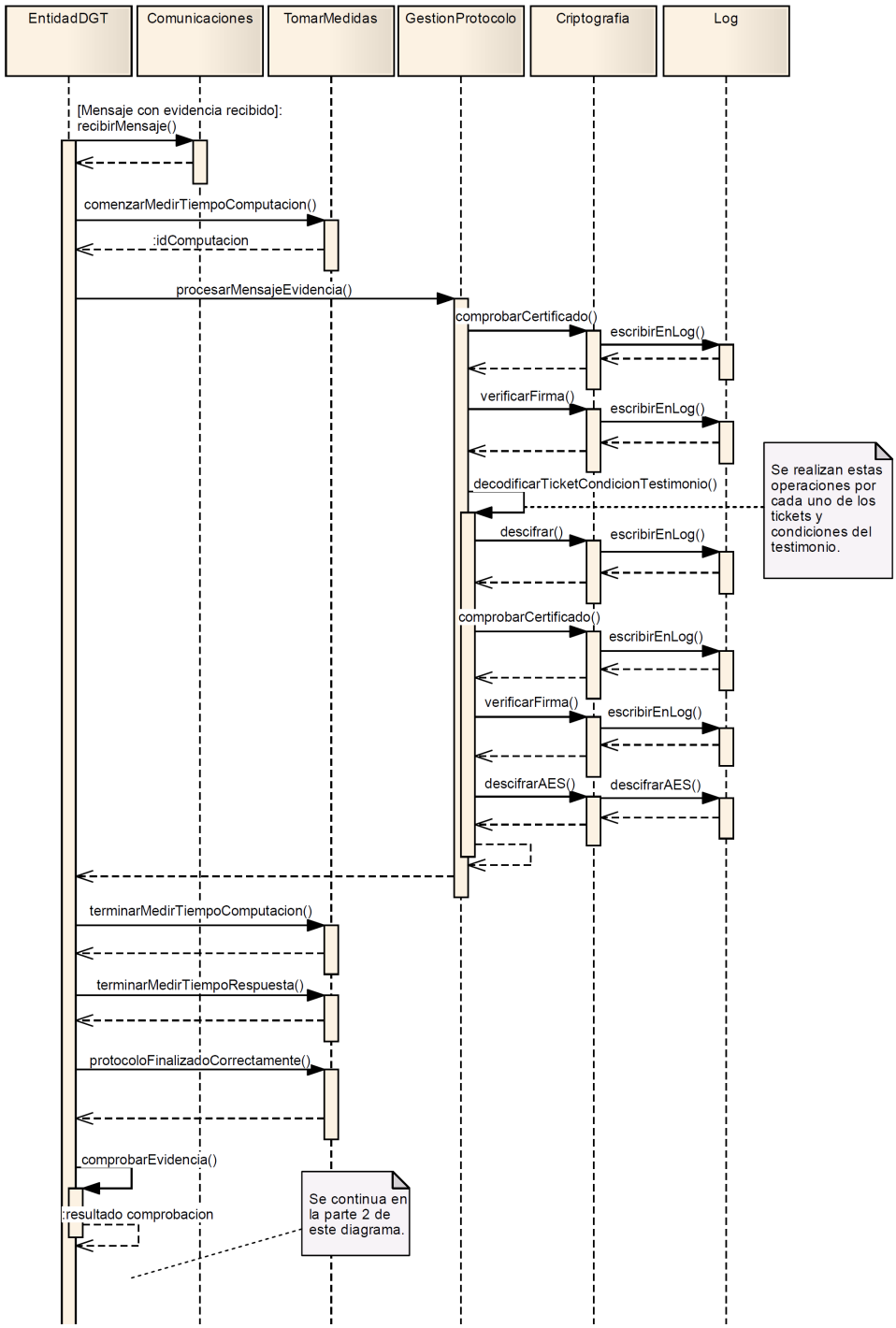

Ilustración 56: diagrama de secuencia 1 de comprobación de una evidencia.

En caso de que el dato de consenso no se corresponda con los testimonios, tal y como ocurre en el caso presente, se debe proceder al envío de un mensaje de revocación de certificado a la AC.

Para ello, en primer lugar se debe verificar si la DGT tiene constancia de la dirección IP de la AC, ya que de lo contrario debe enviar un mensaje a las RSU solicitando la misma. Esta solicitud, tal y como se observa en la Ilustración 57, debe ser creada (*crearMensajePeticionIP_AC*) y enviada a las RSU (*enviarMensaje*). Una vez se recibe la dirección (*recibirMensajeNoBloqueante*) de una RSU, se debe decodificar la misma *(decodificarIP*), con el objetivo de poder utilizarla en el futuro. El motivo por el que se debe utilizar

recibirMensajeNoBloqueante es para evitar el bloqueo indefinido de la DGT si no existe ninguna RSU que responda con la dirección IP.

Una vez la DGT tiene conocimiento de la IP de la AC, se debe crear el mensaje con el identificador del vehículo infractor cuyo certificado se desea revocar (*crearMensajeRevocacionCertificado*), y se manda a la AC (*enviarMensaje*). Se debe indicar que el identificador del vehículo infractor debe ser firmado, por lo que se debe medir el tiempo respectivo de computación e incluir una entrada en el log, y en lo relativo al envío, se debe medir el número de bytes transmitidos y tiempo de envío.

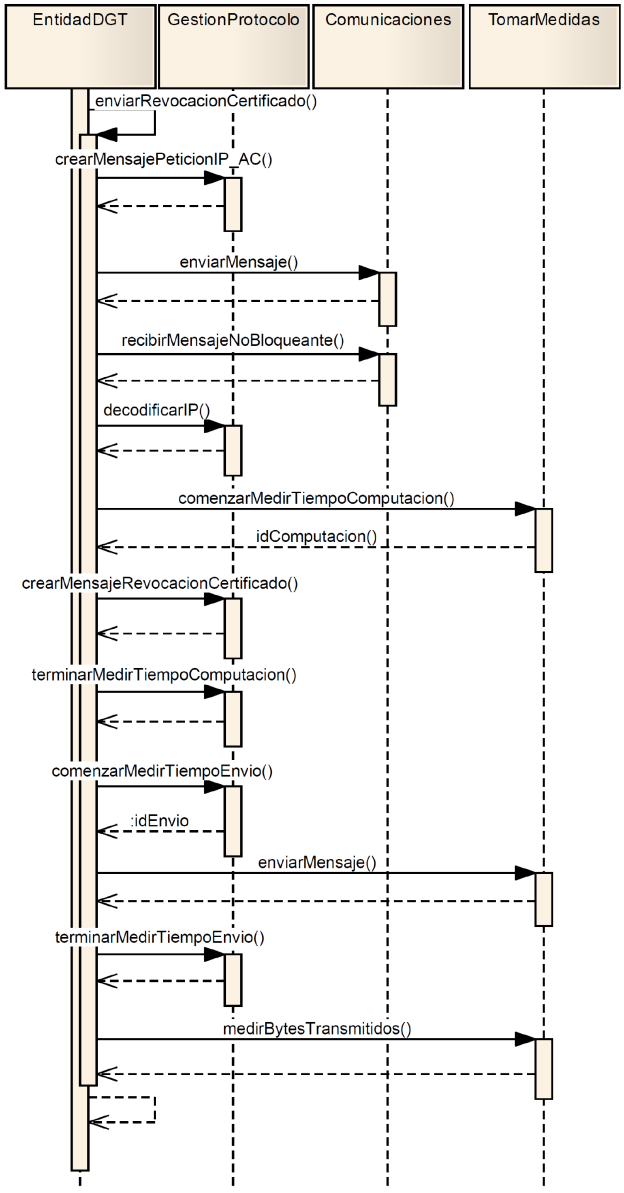

Ilustración 57: diagrama de secuencia 2 de comprobación de una evidencia.

4.1.2.1.1.16. Diagrama de secuencia de envío de estado de DGT.

Como se puede recordar, la DGT envía de manera periódica información de estado a las RSU, con el objetivo de que éstas conozcan la ubicación de la DGT, y con ello, puedan determinar si se encuentran lo suficientemente cercanas para comunicarse.

Teniendo en cuenta que la secuencia lógica entre componentes es similar a la expuesta en el apartado 4.1.2.1.1.9 para el envío del *beacon* de las RSU, por motivos de simplicidad, no se va a incluir el correspondiente diagrama.

4.1.2.1.1.17. Diagrama de secuencia de determinación de RSU próximas a AC.

Al igual que las RSU y la DGT, la AC debe recibir los *beacon* transmitidos por las RSU para determinar cuáles son las RSU cercanas a ella para comunicarse.

Teniendo en cuenta la secuencia lógica para desarrollar esta funcionalidad es similar a la presentada en el apartado 4.1.2.1.1.3, no se va a incluir el correspondiente diagrama de secuencia por simplicidad.

4.1.2.1.1.18. Diagrama de secuencia de revocación de certificado.

En el diagrama mostrado en la Ilustración 58: diagrama de secuencia de revocación de certificado.Ilustración 58, se presenta la interacción necesaria entre los componentes para el desarrollo de la funcionalidad asociada a la recepción de un mensaje de revocación de certificado, inserción del identificador del vehículo infractor en la CRL, y transmisión de la CRL a las RSU próximas a la AC.

En primer lugar, se debe recibir el mensaje de revocación de certificado (*recibirMensaje*), y se debe procesar el mismo para obtener el identificador del vehículo infractor (*procesarMensajeRevocacionCertificado*). También se debe verificar que dicho identificador es correcto y que no se encuentra previamente insertado en la CRL. Además, se debe comprobar que el certificado de la DGT es correcto (*comprobarCertificado*) y haciendo uso del mismo, verificar la firma realizada por la DGT sobre el identificador (*verificarFirma*). Debido a la realización de las operaciones criptográficas de comprobación de certificado y firma, se deben insertar las correspondientes entradas en el log *(escribirEnLog*), y se debe efectuar la medición del tiempo de computación empleado en su realización.

Efectuadas las correspondientes comprobaciones y obtenido el identificador del vehículo, se debe actualizar la CRL insertándolo en ella (*revocarCertificado*), y se debe enviar la misma a las RSU cercanas (*enviarCRL*).

Para ello, se debe codificar la CRL y crear el mensaje en la que va a ser transmitida, adjuntando la firma de la AC sobre el contenido de la CRL (*firmar*) y el tiempo de vida máximo que el mensaje puede circular por la red. Además, considerando la realización de la firma, se debe medir el tiempo de computación correspondiente e incluir una entrada en el log (*escribirEnLog*).

Finalmente, se debe enviar el mensaje a las RSU cercanas (*enviarMensaje*), realizando por cada transmisión la medición del tiempo de envío y número de bytes transmitidos.

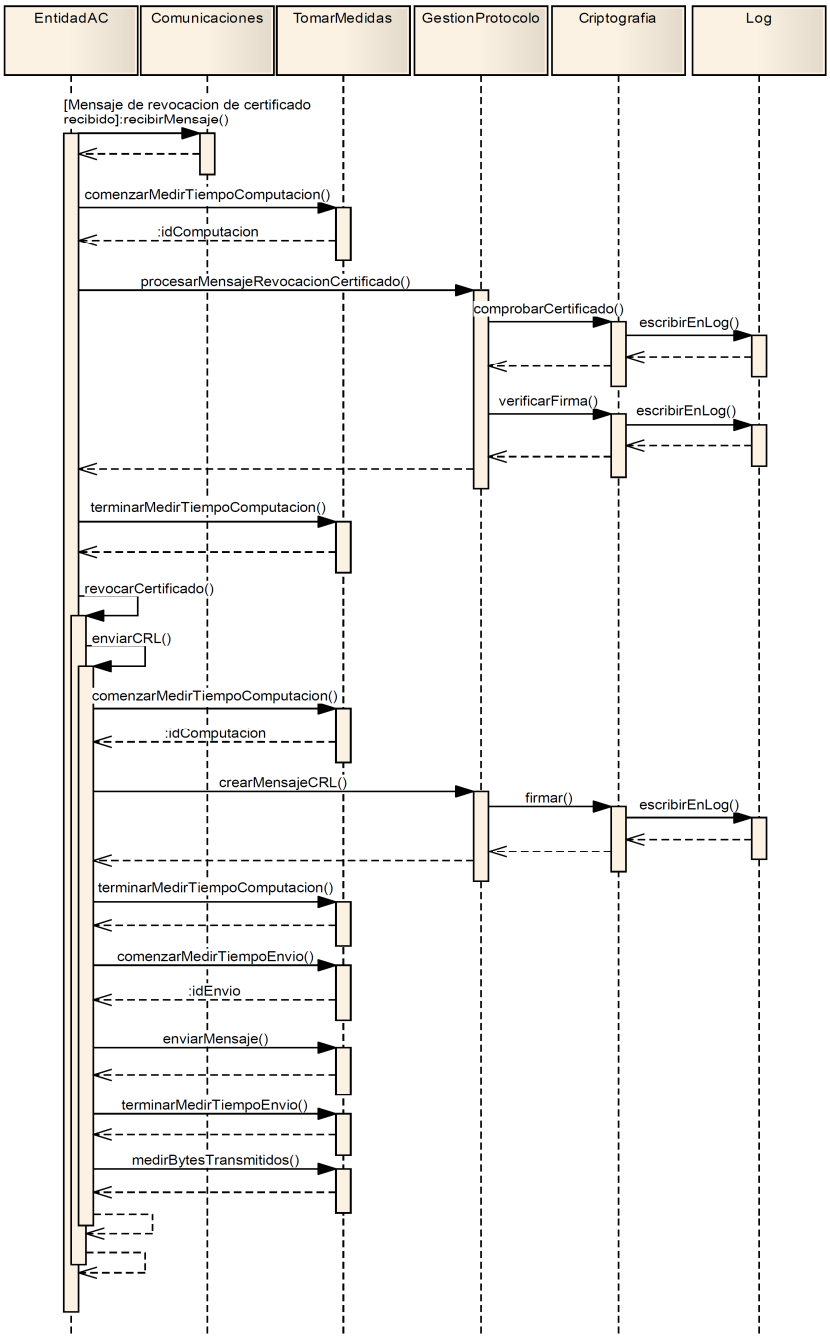

Ilustración 58: diagrama de secuencia de revocación de certificado.

4.1.2.1.1.19. Diagrama de secuencia de reenvío de CRL por AC.

En la Ilustración 59, se presente el diagrama de secuencia encargado del reenvío de la CRL a las RSU cada intervalo de tiempo determinado.

La operación *ejecutarGestionCertificados* es la operación ejecutada por el hilo encargado de la realización de los reenvíos, la cual, expirado el tiempo de reenvío de la CRL, debe ejecutar la operación *enviarCRL*, especificada en el apartado 4.1.2.1.1.18.

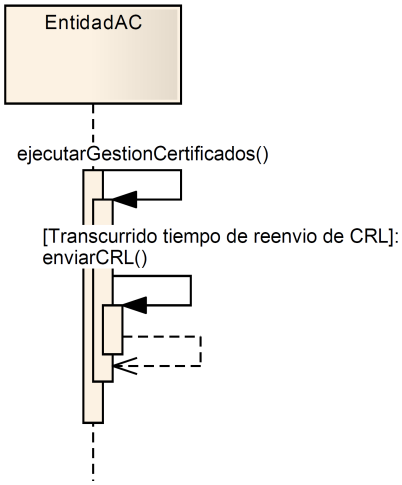

Ilustración 59: diagrama de secuencia de reenvío de CRL por AC.

4.1.2.1.1.20. Diagrama de secuencia de envío de estado de AC.

Al igual que ocurre con la DGT, la AC transmite periódicamente un mensaje de estado indicando su presencia a las RSU. El motivo por el que se hace este envío es para facilitar la inclusión de nuevos mensajes que puedan ser enviados de las RSU a la AC, y para que las RSU puedan obtener la IP de la AC, con el objetivo de proporcionársela a la DGT en caso de solicitud.

No obstante, el diagrama de secuencia asociado a esta funcionalidad no va a ser presentado, debido a que su interacción lógica entre componentes es similar a la presentada en el envío del *beacon* de la RSU (apartado 4.1.2.1.1.9).

4.1.2.2. Diagramas de secuencia asociados a los casos de uso del actor *Usuario sistema representacion indicadores*.

Tal y como se puede verificar en la Ilustración 11 del apartado 3.7, el actor *Usuario sistema representacion indicadores* realiza los siguientes casos de uso:

- *Representar indicadores con sistema reiniciado.*
- *Representar indicadores sin sistema reiniciado.*
- *Almacenar indicadores.*
- *Cargar indicadores.*

Por ello, en los subapartados siguientes se presentan los diagramas de secuencia vinculados a dichos casos de uso.

4.1.2.2.1. Diagramas de secuencia del caso de uso Representar indicadores con sistema reiniciado.

Como se puede recordar, el objetivo del caso de uso *Representar medidas con sistema reiniciado* es el de mostrar en el sistema de representación de indicadores, los indicadores de rendimiento obtenidos de cada una de las distintas entidades que participan en el protocolo, eliminando antes cualquier información previa que pudiese estar representada en el mismo.

En los puntos siguientes, se indican los subapartados en los que se presentan los diagramas de secuencia que representan la funcionalidad asociada a este caso de uso:

- 4.1.2.2.1.2 Diagrama de secuencia de comienzo de representación de indicadores.

4.1.2.2.1.1. 4.1.2.2.1.3 Diagrama de secuencia de parada de representación de indicadores.

-

4.1.2.2.1.2. Diagrama de secuencia de comienzo de representación de indicadores.

En la Ilustración 60: diagrama de secuencia de comienzo de representación de indicadores.Ilustración 60, se presenta el diagrama que muestra la interacción lógica necesaria entre los componentes del sistema de representación de indicadores, para efectuar la funcionalidad relacionada con la representación de indicadores de rendimiento, considerando que el sistema se encuentra en estado parado, y que debe ser eliminada toda la información que pueda estar reflejada de ejecuciones anteriores.

Como se puede comprobar, para iniciar la representación, el usuario debe hacer uso de la operación *comenzarAnalisis* de la interfaz gráfica (*VistaEstadisticas*).

Después, en el Controlador (*ControladorEstadisticas*) se debe verificar que el sistema se encuentra parado, antes de comenzar la representación.

Finalmente, en el Modelo se deben eliminar todos los indicadores almacenados y representados en el sistema de ejecuciones anteriores (*iniciarAnalizador*), y se debe arrancar el hilo encargado de la recepción de los mensajes con los indicadores.

Por tanto, cada vez que se reciba un mensaje con un indicador determinado (*recibirMensaje*), tras comprobar que el formato y campos del mismo son correctos, se debe actualizar la información relacionada con el tipo de dicho indicador (*actualizarEstadisticas*).

Finalmente, con el objetivo de representar los indicadores de rendimiento en la interfaz gráfica a una velocidad adecuada, se debe crear un hilo independiente encargado de realizar dicha labor (*actualizarIndicadores*).

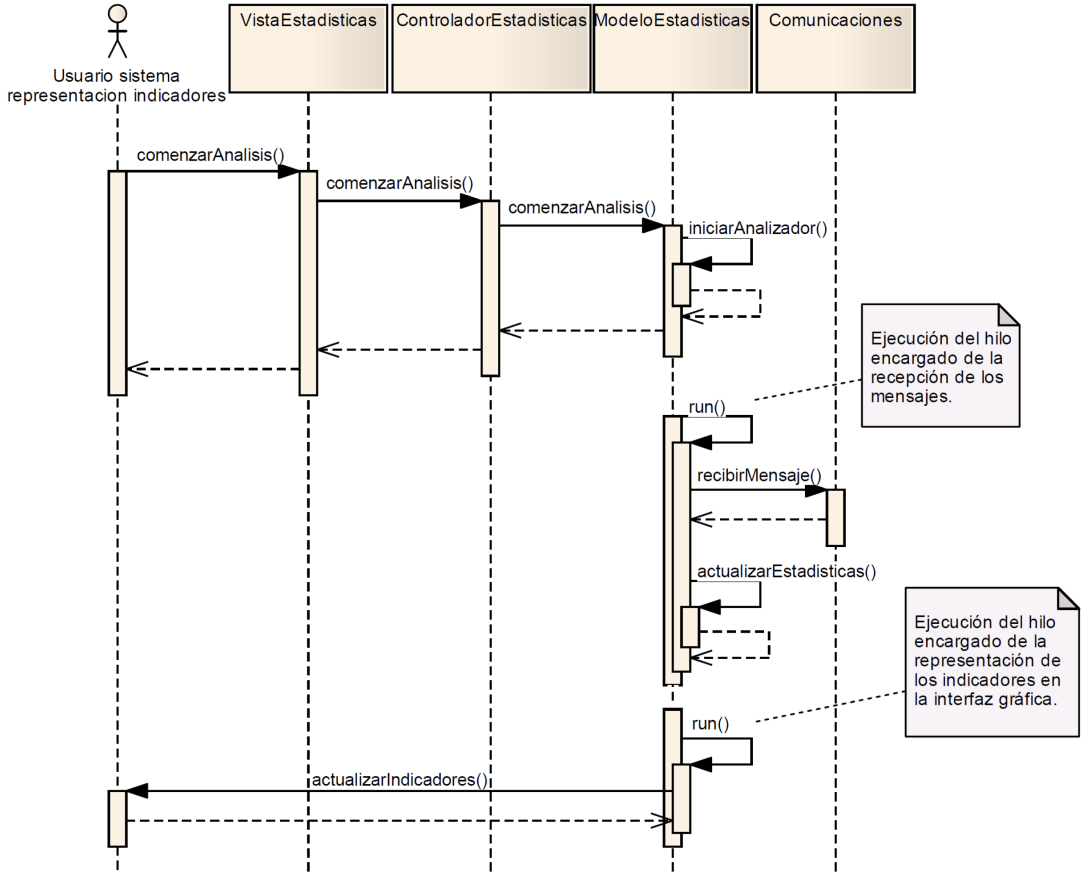

Ilustración 60: diagrama de secuencia de comienzo de representación de indicadores.

4.1.2.2.1.3. Diagrama de secuencia de parada de representación de indicadores.

En el diagrama presentado en la Ilustración 6 se muestra la funcionalidad asociada al escenario alternativo del caso de uso *Representar indicadores con sistema reiniciado*, en la que el sistema se encuentra en funcionamiento reflejando información de rendimiento, y por tanto, tiene que ser previamente parado para iniciar una nueva representación.

Por ello, el usuario debe hacer uso de la operación *pararAnalisis* de la interfaz (*VistaEstadisticas*). Desde ésta, se debe invocar al Controlador (*ControladorEstadisticas*) para verificar que el sistema se encuentra en funcionamiento antes de pararlo, y por último, se debe hacer uso del Modelo (*ModeloEstadisticas*) para terminar la ejecución del hilo encargado de la recepción de los mensajes con los indicadores.

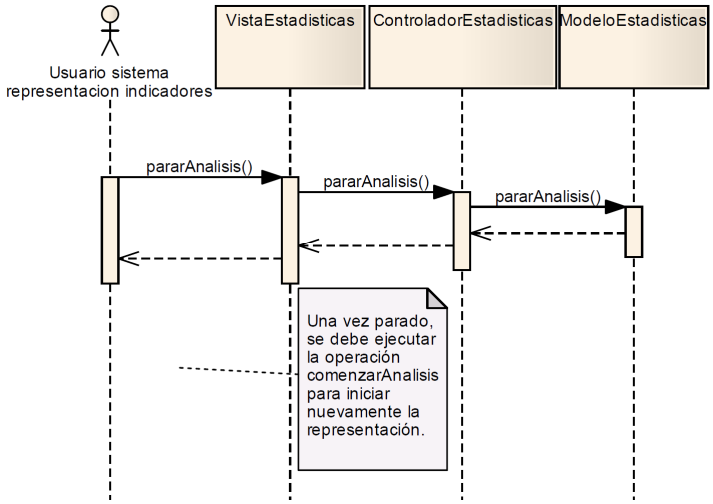

Ilustración 61: diagrama de secuencia de parada de representación de indicadores.

4.1.2.2.2. Diagramas de secuencia del caso de uso Representar indicadores sin sistema reiniciado.

Como se puede apreciar en el apartado 3.7, el objetivo del caso de uso *Representar indicadores sin sistema reiniciado* es la representación de indicadores de rendimiento en el sistema, sin eliminar previamente ninguna información reflejada en el mismo.

Para especificar la funcionalidad relacionada a este caso de uso, en el apartado 4.1.2.2.1 se presenta el diagrama de secuencia relacionado con la continuación de la representación de indicadores de rendimiento.

4.1.2.2.2.1. Diagrama de secuencia de continuación de representación de indicadores.

Como se puede apreciar, en la Ilustración 62 se presenta el diagrama de secuencia correspondiente con la funcionalidad asociada a la representación de indicadores de rendimiento en el sistema, sin necesidad de eliminar información previa que pudiese estar reflejada en el mismo de ejecuciones previas.

Por ello, tal y como se puede observar en el diagrama, el usuario debe hacer uso de la operación *continuarAnalisis* de la interfaz gráfica (*VistaEstadisticas*), la cual debe invocar al Controlador (*ControladorEstadisticas*), para verificar que el sistema se encuentra previamente en estado pausado. Finalmente, se debe hacer uso del Modelo (*ModeloEstadisticas*), para ejecutar el hilo encargado de la recepción de los mensajes con los indicadores. Es necesario indicar que, a partir de la creación de este hilo, la funcionalidad destinada a la recepción y representación de los indicadores se desarrolla de la misma forma que se especificó en el apartado 4.1.2.2.1.2.

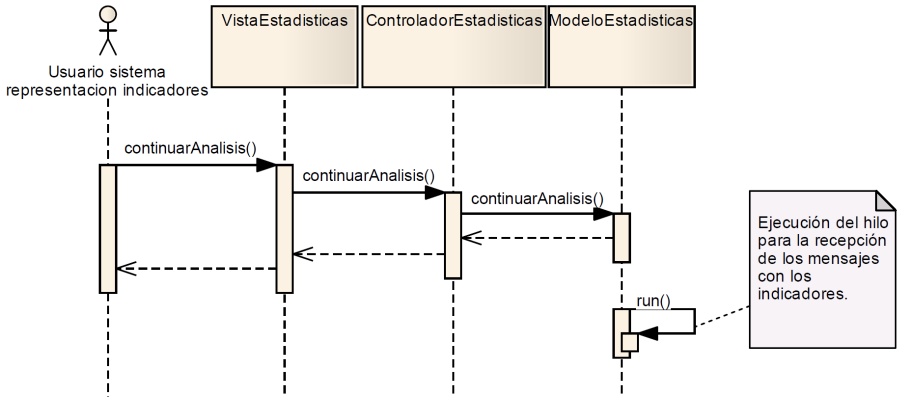

Ilustración 62: diagrama de secuencia de continuación de representación de medidas.

4.1.2.2.3. Diagramas de secuencia del caso de uso Almacenar indicadores.

El objetivo del caso de uso *Almacenar indicadores* es la de almacenar en formato PDF o Excel 2003, la información relacionada con los indicadores de rendimiento.

Por tanto, considerando la existencia de dos tipos de formatos de almacenamiento, se presentan los siguientes diagramas de secuencia.

- 4.1.2.2.3.1 Diagrama de secuencia de almacenamiento de estadísticas en formato PDF.
- 4.1.2.2.3.2 Diagrama de secuencia de almacenamiento de indicadores en formato Excel 2003.

4.1.2.2.3.1. Diagrama de secuencia de almacenamiento de estadísticas en formato PDF.

En la Ilustración 63 se presenta la interacción lógica necesaria entre componentes para realizar el almacenamiento de las estadísticas obtenidas de los indicadores de rendimiento en formato PDF.

Para la realización de dicha funcionalidad, el usuario debe ejecutar la operación *guardarInformacionPDF* de la interfaz gráfica (*VistaEstadisticas*). Desde ésta, se debe hacer uso del Controlador (*ControladorEstadisticas*) para verificar que el sistema se encuentra en estado parado y que la extensión del fichero es correcta, y finalmente, se debe hacer uso de la operación *guardarInformacionPDF* de *ModeloEstadisticas*, la cual utilizando como soporte el componente *GuardarInformacionGraficos*, almacena las estadísticas en formato PDF.

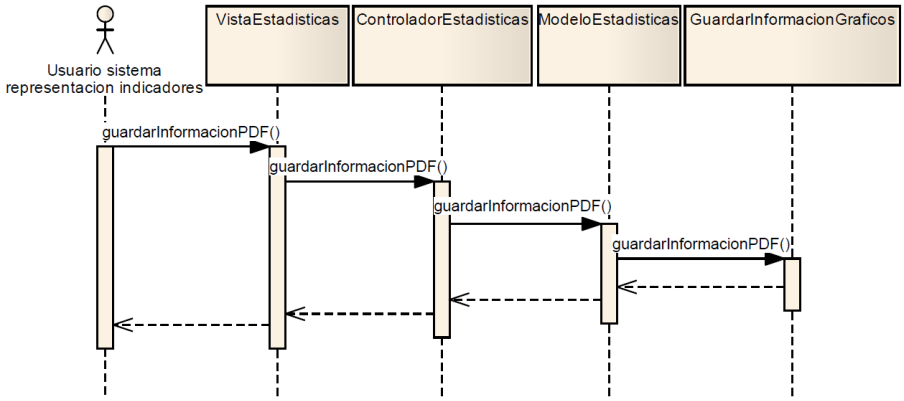

Ilustración 63: diagrama de secuencia de almacenamiento de estadísticas en formato PDF.

4.1.2.2.3.2. Diagrama de secuencia de almacenamiento de indicadores en formato Excel 2003.

La secuencia lógica entre componentes para realizar el almacenamiento de los indicadores en Excel 2003 es similar a la expuesta en el apartado 4.1.2.2.3.1, por lo que no se va a incluir el correspondiente diagrama por motivos de simplicidad. No obstante, se debe considerar el uso del componente *GuardarInformacionDatos*, en lugar de *GuardarInformacionGraficos*.

4.1.2.2.4. Diagramas de secuencia del caso de uso Cargar indicadores.

Como se puede recordar del apartado 3.7, el caso de uso *Cargar indicadores* tiene como fin recuperar los valores de los indicadores de rendimiento almacenados en un fichero Excel 2003, y representarlos en el sistema de representación de los mismos.

Teniendo en cuenta que la funcionalidad que se distingue en este caso de uso es la carga de los indicadores de rendimiento de Excel 2003, en el apartado 4.1.2.2.4.1 se presenta el diagrama asociado a la misma.

4.1.2.2.4.1. Diagrama de secuencia de carga de indicadores de formato Excel 2003.

El diagrama de secuencia lógico asociado a la carga de los indicadores de Excel 2003 se muestra en la Ilustración 64.

Como se puede verificar, el usuario debe ejecutar la operación *cargarInformacion* de la interfaz (*VistaEstadisticas*), desde la cual se debe hacer uso del Controlador (*ControladorEstadisticas*) para comprobar que el sistema se encuentra en estado parado y que la extensión del fichero Excel es correcta. Por último, se debe recuperar la información de los indicadores de rendimiento del fichero utilizando como soporte el componente *CargarInformacionDatos* (*cargarInformacion*), y finalmente, se debe representar dicha información en la interfaz (*actualizarIndicadores*).

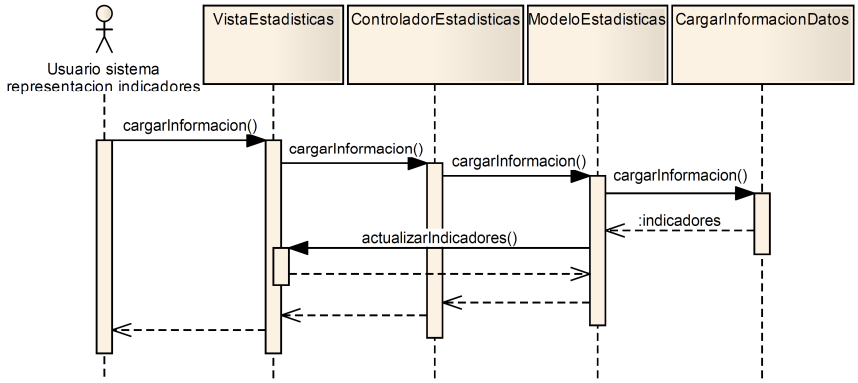

Ilustración 64: diagrama de secuencia de carga de indicadores de formato Excel 2003.

5. Implementación y pruebas del software.

En esta sección se describen una serie de decisiones tomadas durante la fase de implementación, y se presentan los resultados obtenidos de las pruebas de aceptación. En los párrafos siguientes se describe de forma más detallada el contenido de esta sección.

En primer lugar, se especifican una serie de decisiones sobre la implementación del software que no han sido tratadas durante las fases de análisis y diseño anteriores.

Finalmente, se presentan los resultados obtenidos de la realización de las pruebas de aceptación cuyo diseño fue especificado en el apartado 3.9.

5.1. Decisiones de implementación.

En este apartado se especifican una serie de decisiones tomadas sobre la implementación del sistema.

En primer lugar, es necesario comentar que durante el estudio realizado del simulador NCTUns [Referencias-1] en la primera etapa del análisis, fue determinada una restricción que consistía en que el simulador no disponía de medios para determinar cuando dos entidades, por ejemplo dos vehículos, se encontraban lo suficientemente cercanas para comunicarse. Con el fin de solventar esta restricción, en el diseño se consideró que las entidades enviasen su ubicación en el entorno simulado, y en función de la distancia a la que se encontraran, determinar si podían establecer comunicación. Posteriormente, durante la etapa de implementación, se encontró una opción en el simulador que permitía determinar la capacidad de transmisión de un nodo, lo que implica que no existiese la restricción que fue determinada en un primer momento. Por este motivo, esta restricción fue eliminada de las restricciones especificadas en el estudio del simulador. No obstante, se ha implementado la solución siguiendo el diseño establecido, debido a que se considera más intuitivo para un usuario que quiera hacer evaluaciones del protocolo, que establezca la distancia máxima de comunicación entre las entidades de los ficheros de configuración de los que hacen uso las mismas, y de esta manera, no tenga necesidad de conocer en profundidad todas las posibilidades que ofrece NCTUns [Referencias-1] para la realización de las simulaciones.

Otra decisión de implementación que no fue considerada en el diseño inicial fue la necesidad de la DGT de conocer la IP de la AC, para realizar el envío del mensaje de revocación de certificado. En un primer momento se pensó en incluir la IP como variable de configuración de la DGT. Sin embargo, eso implicaba que el usuario tuviese que cambiar dicho valor cada vez que crease un escenario nuevo, o cada vez que el simulador le asignase otro valor a la IP de la AC. Por tanto, por no complicar la labor del usuario que quiera realizar simulaciones de las entidades, se actualizó el diseño especificando la solución descrita en dicho apartado (que la DGT solicite a las RSU la IP de la AC).

Con el objetivo de que las entidades puedan realizar las distintas operaciones criptográficas que necesitan, deben disponer de un certificado de clave pública y su correspondiente clave privada. Teniendo en cuenta que no es objetivo de las entidades crear esos elementos criptográficos, sino que deben disponer de los mismos desde el comienzo de su ejecución, se deben generar de forma externa haciendo uso de los comandos proporcionados por OpenSSL [Referencias-3]. No

obstante, si se tiene un número muy elevado de entidades, el tener que estar generando un certificado y clave por cada una de ellas supone un consumo de tiempo muy elevado. Por tanto, para evitar esta situación, y considerando que el objetivo fundamental es la evaluación del rendimiento del entorno vehicular en el que participan las entidades, no importa que éstas no dispongan de certificado propio, por lo que se genera un certificado y clave por cada rol (por ejemplo, un certificado y clave para todas las entidades RSU), y de esta manera, se utilizan siempre los mismos recursos criptográficos con independencia del número de entidades que se dispongan.

Finalmente, se tuvo que tomar una decisión de implementación importante para solucionar un problema producido al realizar la comprobación de la validez de los certificados de las entidades. El problema consistía en que el API criptográfico de OpenSSL [Referencias-3] proporciona una función que sirve de soporte para la realización de dicha comprobación. Dicha función se encarga de comprobar si un certificado ha sido firmado con la clave privada correspondiente con la clave pública de un determinado certificado, si se encuentra dentro de su periodo de validez, si no ha expirado, etc. No obstante, al realizar la comprobación de los certificados de las entidades con dicha función, el resultado que siempre proporcionaba era incorrecto. Después de probar muchas posibilidades, se hizo la correspondiente verificación utilizando el comando para la comprobación de certificados de OpenSSL [Referencias-3], y con dicho comando, se verificó que la correspondiente comprobación daba un resultado correcto. Por tanto, con el objetivo de solucionar el problema de la comprobación de certificados, cada vez que una entidad debe ejecutar dicha operación, debe hacer uso del comando correspondiente de OpenSSL [Referencias-3].

5.2. Resultado de las pruebas de aceptación.

En este apartado se especifican los resultados obtenidos de las pruebas de aceptación del sistema, indicando para cada una de ellas si se ha superado la prueba, y en caso de no ser así, los motivos por los que no ha sido superada.

En primer lugar, en el subapartado siguiente se muestra la plantilla empleada para la especificación del resultado de las pruebas de aceptación.

5.2.1. Formato para la especificación de los resultados de las pruebas de aceptación.

En la Tabla 23, se presenta la plantilla utilizada para especificar el resultado de las pruebas de aceptación.

RESULTADOS PRUEBAS DE ACEPTACIÓN	
Id	**Resultado**

Tabla 23: catálogo Resultados Pruebas de Aceptación.

- Id: identificador único asociado a la prueba de aceptación, el cual debe ser igual a su correspondiente especificado en el diseño de las pruebas.

- Resultado: resultado obtenido tras la realización de la prueba. El valor de este campo es "Éxito" si la salida de la prueba se corresponde con la salida especificada en el diseño de la misma, y "Fallo" en caso contrario. Además, en caso de que la prueba no se supere, se deben indicar los motivos del fracaso, con el objetivo de aplicar los medios necesarios para su solución.

5.2.2. Especificación de los resultados de las pruebas de aceptación.

En este apartado se realiza la especificación de los resultados de las pruebas de aceptación.

Teniendo en cuenta que el diseño de las pruebas de aceptación del apartado 3.9 fue realizado atendiendo a las dos partes en las que se distingue el sistema, la extensión de NCTUns [Referencias-1] para la simulación del entorno de infraestructura del protocolo EVIGEN [Bibliografía-1], y el sistema de representación de indicadores, en los subapartados siguientes se presenta la especificación de cada una de las partes.

5.2.2.1. Especificación de los resultados de las pruebas de aceptación de la extensión de NCTUns para la simulación del entorno de infraestructura del protocolo EVIGEN.

En la Tabla 24 se presentan los resultados de las pruebas de aceptación de la extensión de NCTUNs [Referencias-1] para la simulación del entorno de infraestructura de EVIGEN [Bibliografía-1].

RESULTADOS PRUEBAS DE ACEPTACIÓN	
Id	**Resultado**
P-E-01	Éxito.
P-E-02	Éxito.
P-E-03	Éxito.
P-E-04	Éxito.
P-E-05	Éxito.
P-E-06	Éxito.
P-E-07	Éxito.
P-E-08	Éxito.
P-E-09	Éxito.
P-E-10	Éxito.
P-E-11	Éxito.
P-E-12	Éxito.
P-E-13	Éxito.
P-E-14	Éxito.
P-E-15	Éxito.
P-E-16	Éxito.
P-E-17	Éxito.
P-E-18	Éxito.
P-E-19	Éxito.
P-E-21	Éxito.
P-E-20	Éxito.
P-E-22	Éxito.

Tabla 24: resultado de las pruebas de aceptación de la simulación del entorno infraestructura de EVIGEN.

5.2.2.2. Especificación de los resultados de las pruebas de aceptación del sistema de representación de indicadores.

Los resultados de las pruebas de aceptación del sistema de representación de indicadores se presenta en la Tabla 25.

RESULTADOS PRUEBAS DE ACEPTACIÓN	
Id	**Resultado**
P-I-01	Éxito.
P-I-02	Éxito.
P-I-03	Éxito.
P-I-04	Éxito.
P-I-05	Éxito.
P-I-06	Éxito.
P-I-07	Éxito.
P-I-08	Éxito.
P-I-09	Éxito.
P-I-10	Éxito.

Tabla 25: resultado de las pruebas de aceptación del sistema de representación de indicadores.

6. Conclusiones y líneas futuras.

Para finalizar el Proyecto, en esta sección se exponen las principales conclusiones que se pueden obtener del desarrollo del mismo, así como las principales dificultades encontradas en el mismo, y finalmente, se describen una serie de líneas futuras.

6.1. Conclusiones sobre el Proyecto.

En este apartado, se presentan las conclusiones consideradas como las más relevantes sobre el desarrollo del Proyecto, y posteriormente, se indican una serie de aspectos a tener en cuenta sobre la dificultad presente en el mismo.

6.1.1. Desarrollo del Proyecto.

En este Proyecto se ha ampliado la funcionalidad del simulador NCTUns 5.0 [Referencias-1] de manera que se pudiese simular en el mismo el comportamiento de las entidades que componen el entorno de infraestructura del protocolo EVIGEN [Bibliografía-1], y se ha desarrollado un sistema capaz de representar gráficamente una serie de indicadores de rendimiento obtenidos del protocolo.

El motivo por el que se ha realizado este desarrollo, y por tanto, la idea sobre la que se ha sustentado el Proyecto, es debido a la necesidad de disponer de herramientas que permitiesen realizar un estudio de EVIGEN [Bibliografía-1] en un entorno ficticio con características muy similares a las de las redes vehiculares reales, con el objetivo de analizar su funcionamiento y rendimiento sobre las mismas de manera precisa, y de esta manera, determinar la viabilidad de su implantación. Con ello se consigue evitar grandes pérdidas, especialmente en el ámbito económico, en caso de que el protocolo no satisfaga las necesidades para las que ha sido desarrollado, ya que los costes de implantación en este tipo de entornos son muy elevados. Por tanto, considerando que NCTUns [Referencias-1], al igual que el resto de simuladores de redes vehiculares, no dispone de capacidad para poder definir protocolos que puedan ser simulados en ellos, ni medios para evaluar su rendimiento, se ha tenido que ampliar la funcionalidad del mismo para conseguir los objetivos propuestos.

Además, considerando las amenazas presentes en este tipo de escenarios, se ha desarrollado un módulo que ofrece una serie de operaciones criptográficas que pueden ser utilizadas para la incorporación de servicios de seguridad a los diferentes protocolos que puedan ser simulados en NCTUns [Referencias-1].

Por tanto, considerando lo comentado en los párrafos anteriores, se puede determinar que se han cumplido con los objetivos del Proyecto presentados en el apartado 1.1.

Para asegurar el correcto cumplimiento de dichos objetivos, se ha realizado un Proyecto desarrollado mediante técnicas propias de Ingeniería del Software, que incluye las etapas de planificación, análisis, diseño, implementación y pruebas.

En la fase de planificación resulta interesante comentar la realización de la planificación y presupuesto inicial, y el posterior análisis de la desviación cometida con respecto al resultado real obtenido en el Proyecto.

En la fase de análisis cabe destacar el estudio del simulador, el análisis de seguridad, la selección de la tecnología necesaria para la implementación, la

especificación de los requisitos de software y el diseño de las pruebas de aceptación.

De la fase de diseño resulta importante señalar la realización de un diseño basado en componentes, los cuales puedan, en la medida de lo posible, ser reutilizados en proyectos posteriores. Además, se ha hecho uso de ciertos patrones de diseño, como *Strategy* [Bibliografía-3], para facilitar el mantenimiento y extensibilidad del diseño. Por último, otro de los criterios de diseño ha sido la eficiencia en la implementación, habida cuenta de que se trata de un aspecto fundamental en el ámbito de la simulación.

En lo que respecta a la fase de implementación del software, éste ha sido desarrollado según el diseño establecido, de manera que se cumpliesen con los requisitos de software especificados.

Por tanto, teniendo en cuenta que este Proyecto es desarrollado en solitario, se ha aprendido a realizar un trabajo de ingeniería en el que se desarrollan las principales fases de éste, y a tener capacidad suficiente para poder afrontar y solucionar los principales problemas y dificultades presentes durante el mismo.

Otro aspecto aprendido de la realización de este proyecto son las técnicas básicas de investigación que han sido necesarias para comprender y ampliar el simulador, y además, a ser consciente de la complejidad de integrar una solución nueva en un sistema cuya implementación se desconoce totalmente.

Finalmente, también se aprende a ser consciente de la importancia que tiene realizar un riguroso estudio previo del funcionamiento y rendimiento de un sistema cuya implantación puede suponer fuertes pérdidas económicas, como el del protocolo EVIGEN [Bibliografía-1] en el ámbito vehicular.

6.1.2. Dificultad del Proyecto.

En lo que respecta a la dificultad del Proyecto, en primer lugar cabe mencionar la complejidad presente en la realización de todo Proyecto de Ingeniería del Software, debido a la gran cantidad de aspectos que se deben considerar en el mismo. Además, a esta dificultad se añade el hecho de haber tenido que realizar el Proyecto en solitario, ya que con un equipo de trabajo adecuado, el resultado de su desarrollo posiblemente hubiese presentado una mayor calidad.

Otra dificultad presente en este proyecto es la necesidad de tener que utilizar como base el simulador NCTUns [Referencias-1], ya que se ha necesitado realizar un estudio previo del funcionamiento del mismo antes de proceder a desarrollar una solución que permita ampliar su funcionalidad. Además, teniendo en cuenta las restricciones que presenta el mismo, las cuales han afectado a la implementación del software, se han tenido que tomar las decisiones necesarias que solventasen las mismas.

En lo que respecta a la implementación del sistema, se considera que ésta presenta una complejidad considerable ya que se han tenido que crear dos sistemas independientes (la implementación del entorno infraestructura de EVIGEN [Bibliografía-1] y el sistema de representación de indicadores), la cantidad de funcionalidad a desarrollar ha sido adecuada, se ha hecho uso de una gran cantidad de comunicaciones, se ha tenido que utilizar tecnología desconocida hasta el momento como la librería criptográfica de OpenSSL [Referencias-3], etc.

Además, debido a la integración con el simulador, encontrar los errores cometidos en la implementación y solucionarlos, ha presentado una complejidad mayor que la existente en sistemas tradicionales.

Finalmente, otro aspecto importante es que, teniendo en cuenta que este Proyecto ha tenido que ser integrado junto con otro desarrollado en paralelo [Bibliografía-2], se ha necesitado de cierta coordinación entre las partes para conseguir este propósito.

6.2. Líneas futuras.

En los párrafos siguientes, se describen algunos aspectos que no forman parte del presente Proyecto, los cuales pueden ser considerados en un futuro para ampliar el trabajo realizado en el mismo.

En primer lugar, teniendo en cuenta que en el sistema desarrollado únicamente se detectan infracciones por exceso de velocidad, se puede ampliar el mismo incluyendo nuevos tipos de infracciones, como por ejemplo, circular a una velocidad inferior al mínimo establecido.

Además, en lo que respecta a la detección de las infracciones, considerando que en el sistema implementado se basa en la obtención y comprobación del *beacon* transmitido por los vehículos, se puede implementar algún mecanismo que permita la realización de la misma de manera más fiable, como por ejemplo simular la existencia de sensores en la vía.

También, otro punto a tener en cuenta en una ampliación futura, extraído del análisis de seguridad, es que las RSU transmitan de manera periódica un *beacon* firmado, con el objetivo de que se pueda garantizar su integridad y no repudio.

Otro aspecto a considerar en el futuro es la modificación del funcionamiento de NCTUns [Referencias-1], con el objetivo de que se puedan realizar las simulaciones en un entorno distribuido y no sólo sobre la máquina local, de manera que cada entidad se ejecutase en una máquina distinta, permitiendo una mejor escalabilidad del sistema, y además, consiguiendo que la información de rendimiento obtenida sea mucho más precisa.

Por último, comprobando la existencia de una nueva versión del simulador NCTUns [Referencias-1], la versión 6.0, otra posible línea a ser tratada en el futuro es la migración de la implementación realizada en este Proyecto sobre la versión 5.0 a la nueva versión, ya que posiblemente esta versión presente mejores prestaciones que la anterior, y además, se pueden haber solucionado algunas de las limitaciones encontradas en el simulador.

7. Bibliografía y referencias.

En esta sección se incluye la lista con la bibliografía consultada, y la lista con los recursos a los que se ha hecho referencias, a lo largo del Proyecto, indicando únicamente aquellos especificados en el presente documento.

7.1. Bibliografía.

La bibliografía utilizada en este proyecto es la expuesta a continuación:

1. DE FUENTES, JOSÉ MARÍA; GONZÁLEZ-TABLAS, ANA ISABEL; RIBAGORDA, ARTURO; RAMOS, BENJAMÍN. *Protocolo de creación de evidencias en entornos vehiculares.* Congreso Iberoamericano de Seguridad de la Información (CIBSI), Montevideo (Uruguay), 2009.
2. GONZÁLEZ MANZANO, LORENA. *Extensión de NCTUns 5.0 para simular el entorno inalámbrico y desarrollo del visor de mensajes intercambios para EVIGEN*. Proyecto de Fin de Carrera, Universidad Carlos III de Madrid, 2010. (pendiente de publicación)
3. GAMMA, ERICH; HELM, RICHARD; JOHNSON, RALPH; VLISSIDES, JOHN**.** *Patrones de Diseño.* s.l. : Addison-Wesley, 1995.
4. GÉNOVA, GONZALO. Apuntes Ingeniería del Software I Universidad Carlos III de Madrid - Modelado de casos de uso. 2010.

7.2. Referencias.

La lista con los recursos a los que se ha hecho referencia durante el proyecto son los siguientes:

1. WANG, S.Y. *NCTUns Network Simulator and Emulator.* [En línea] http://nsl.csie.nctu.edu.tw/nctuns.html.
2. FEDORA. *Proyecto Fedora*. [En línea] http://fedoraproject.org/.
3. OPENSSL. *OpenSSL: The Open Source toolkit for SSL/TLS.* [En línea] http://www.openssl.org/.
4. DAI, WEI. *Crypto++ Library 5.6.1 - a Free C++ Class Library of Cryptography Schemens.* [En línea] http://www.cryptopp.com/.
5. Botan. [En línea] http://botan.randombit.net/.
6. OBJECT MANAGEMENT GROUP. *OMG's CORBA WebSite*. [En línea] http://www.corba.org/.
7. *gSoap: SOAP C++*. [En línea] http://www.cs.fsu.edu/~engelen/soap.html.
8 APACHE. *WebServices-Axis.* [En línea] http://ws.apache.org/axis/.
9. MYSQL. *MySQL::The world's most popular open source database.* [En línea] http://www.mysql.com/.
10. GTK+. *GTK+*. [En línea] http://www.gtk.org/.
11. ORACLE. *Java 2D API.* [En línea] http://java.sun.com/products/java-media/2D/index.jsp.
12. VIKLUND, ANDREAS. *JFreeChart.* [En línea] http://www.jfree.org/jfreechart/.
13. *iText.* [En línea] http://itextpdf.com/.
14. KHAN, ANDY. *JExcelApi.* [En línea] http://jexcelapi.sourceforge.net/.
15. MICROSOFT. *Microsoft Visual Studio 2010.* [En línea] http://www.microsoft.com/spain/visualstudio.

16. NOVELL. *Mono.* [En línea] http://www.mono-project.com/Main_Page.
17. NETBEANS. *NetBeans.* [En línea] http://netbeans.org/.
18. EUROPEAN SPACE AGENCY. *ESA.* [En línea] http://www.esa.int/esaCP/index.html.